ADAPTIVE LOGIC TECHNIQUES AND DECISION-MAKING

Milton Leroy Senft

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ADAPTIVE LOGIC TECHNIQUES AND
DECISION-MAKING

by

Milton Leroy Senft

March 1975

Thesis Advisor: R. Panholzer

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Adaptive Logic Techniques and Decision-Making		5. TYPE OF REPORT & PERIOD COVERED Electrical Engineer; March 1975 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Milton Leroy Senft		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE March 1975 13. NUMBER OF PAGES 97
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Adaptive Logic
Artifidal Intelligence
Chess
Heuristic Search

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper represents a study of the problems and techniques of decision-making by computer. Current methods of machine decision-making are described and analyzed. The concepts of artificial intelligence, machine learning, heuristic search and adaptive logic are introduced. Two methods of training adaptive logic elements are described. As an example of decision-making by computer, two computer

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block #20 continued

programs which together develop parameters and make decisions in chess are presented.

Adaptive Logic Techniques and Decision-Making

by

Milton Leroy Senft
Lieutenant, United States Navy
B. S., Purdue University, 1967

Submitted in partial fulfillment of the
requirements for the degree of

ELECTRICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
March 1975

ABSTRACT

This paper represents a study of the problems and techniques of decision-making by computer. Current methods of machine decision-making are described and analyzed. The concepts of artificial intelligence, machine learning, heuristic search and adaptive logic are introduced. Two methods of training adaptive logic elements are described. As an example of decision-making by computer, two computer programs which together develop parameters and make decisions in chess are presented.

TABLE OF CONTENTS

LIST OF DRAWINGS

I. DECISION-MAKING AND ARTIFICIAL INTELLIGENCE

A. PROBLEM REPRESENTATION

There is a fairly general and important class of problems which can be represented in the form of a tree structure [1]. This class of problems includes applications such as problems in chess, geometry and indefinite integration. The tree structure of this class of problems is formed as follows: The problem is represented in a state space such that the initial state is the problem state and the solution is the goal state. In order to reach the goal state it may be necessary to pass through one or more sub-goal states. If the states are represented as nodes, and all possible movements from one node to another are represented by branches, then the problem description is a tree structure as shown in Figure 1. The nodes at level L+1 generated from a node at level L are known as first successor nodes. Thus in Figure 1 nodes 11, 12 and 13 are first successors to node 01.

Generation of all possible successor nodes from any given node in a tree is known as complete node expansion. One approach to general problem solving is to expand the problem node into a set of first successor nodes, then expand each of the successor nodes, and so on until either the goal state - node - is reached or all possibilities are exhausted. Implicit in this technique is a test conducted at each node to determine whether the goal state has in fact been reached.

TREE STRUCTURE

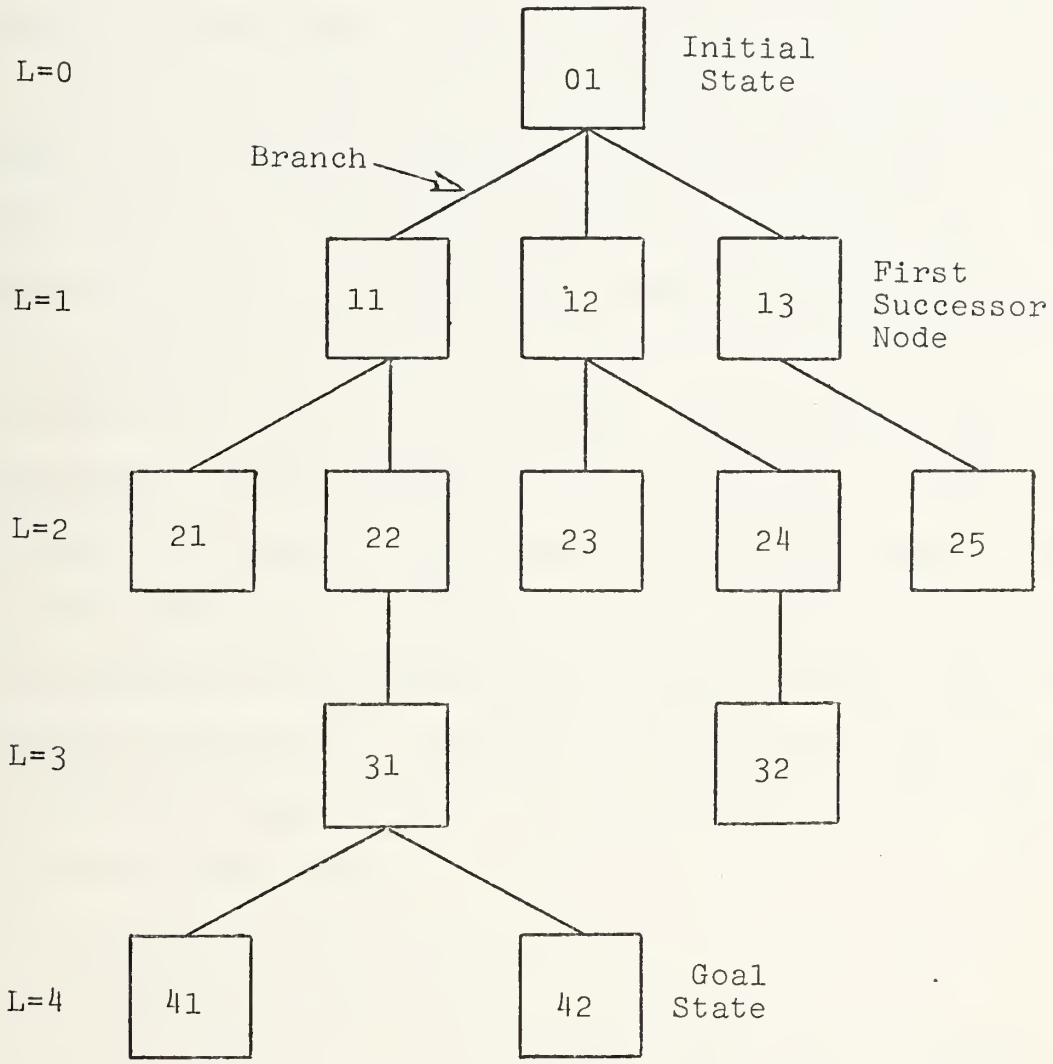


Figure 1.

It is not difficult to see that a general tree structure expands exponentially. That is, if each node has n successor nodes, then the number of nodes at each level L - with L=0 for the initial node - can be expressed as n^L.

For problems with n or L - or both - large, exhaustive search of the tree as proposed above becomes impossible, even if the search is carried out by a high-speed computer. The problem is said to suffer from combinatorial explosion.

Problems with very large search trees form a set of very difficult and interesting problems, particularly from the standpoint of programming their solution on a computer. Many of these large problems represent areas where human beings can cope very well. Examples include theorem-proving, sentence-parsing, the recognition of cursive script, chess and language translation. Research on the solution of this kind of problem by computer has come to be known as the study of Artificial Intelligence.

B. ARTIFICIAL INTELLIGENCE

As an example of an Artificial Intelligence problem, consider the game of chess. Chess can be represented by a tree with each board position a node and each possible move a branch. As in all two-person competitive game trees, since each person makes a decision only at every alternate level, the term ply has come into use as a descriptor of a player's decision levels. Each level therefore represents ½ ply, and each player will make a decision at ply n from his starting node, where n=1,2,3,...

Each node in chess typically generates around 30 successors [2]. The complete game tree in chess is, for all practical purposes, infinite. Yet men play chess, and some men play it very well.

It is known [3] that even chess grand masters search fewer than 100 nodes before arriving at a decision for a move. Modern chess programs search many thousands of nodes, and yet their most successful level of play is only at the high amateur level [4].

The crucial aspect is that the grand master searches only a small portion - indeed a fraction too small to be measured - of the tree, but he somehow selects the most productive branches on which to carry out his limited search. The process of eliminating branches from consideration for further search is known as pruning, and the process whereby one branch is selected for action is known as decision-making.

The main purposes of the study of Artificial Intelligence are to understand human intelligence, to find rules of decision-making which can be used by a computer, and to use machine intelligence to solve difficult problems [5].

1. Human Decision-Making

As explained above, the process whereby one branch from a node is selected from others for action is known as decision-making. Human beings make decisions in complex ways which are little understood.

One branch of Artificial Intelligence is interested in determining how the human brain makes decisions. Researchers who study man's decision-making processes make use

of a technique known as protocols. In using the protocol method the researcher observes the subject thinking aloud while trying to solve a problem. In chess, this method has extended to study of the eye movements of grand masters while contemplating play. From the study of protocols, researchers construct a computer program model of the problem-solving techniques displayed, and note how the results of their program differ from that of the human subject.

2. Machine Decision-Making

The other branch of Artificial Intelligence is interested only in obtaining intelligent behavior from a computer. A researcher from this school is not concerned whether the structure of his program in any way resembles the human decision-making process. He is concerned with results.

One method used by researchers of this school is that of heuristic programming. A heuristic [6] is a rule of thumb, strategy or trick which is used to ehance the decision-making process of a computer program. Related to the word "eureka," - from the Greek heuriskein, serving to discover - a heuristic is a technique used by the computer programmer, not by the computer.

In order to attack Artificial Intelligence problems by computer, the problem must first be structured in such a way that the computer can deal with it. Since computers can only perform a strictly defined and limited set of operations, it is the task of the computer program to present the problem as a series of operations which the machine can carry out.

The programmer uses heuristics when he translates a complex problem into a series of machine operations.

An example of a heuristic used in chess is a method to determine control of the center of the board. A programmer could use the heuristic that occupation of a majority of the center four squares on the board by, say, White represents control of the center by White. Based on this heuristic, the programmer would instruct the machine to determine which player - if any - has more pieces on the center four squares. If the heuristic is a good one, then the answer computed by the machine will be equivalent to the determination of center control by a grand master.

In most problems of Artificial Intelligence, the programmer will use many such heuristics in an attempt to enable the computer to evaluate a node and to decide which branch from that node is most likely to lead towards the goal state. The combination of a number of heuristically programmed operations used to evaluate a node is known as an evaluation function.

The basic idea of heuristic programming is to use information derived from the problem to guide the search through the problem nodes in such a way as to limit the number of nodes searched while still successfully finding the goal state (node).

Heuristic search may be either breadth-first or depth first. If all first successors of each node are searched before more nodes are generated, then the resulting search is called a breadth-first search, as shown in Figure

2. If the search proceeds from one of the first successors to following levels before visiting other nodes at the early levels, it is termed a depth-first search, as shown in Figure 3. Roughly speaking, a breadth-first procedure generates from the top, while a depth-first procedure generates from the left.

There are two general methods of tree pruning widely used in heuristic programming. The first involves depth-first search wherein the last node generated - for example node 7 at level 3 in Figure 3 - is assigned a value based on the computer's evaluation function. Based on this value, the predecessor node at level 2 (node 2) is assigned a value, and so on until the initial node is reached. This process is called "backing-up" the value of one node to its predecessors, and if, based on the value backed-up to the initial node, this whole branch (nodes 1 to 7) is eliminated from further consideration then it is said that the tree has been pruned by backward pruning.

The second method of tree pruning is to use information available at a node to eliminate from consideration some branches emanating from that node. This technique is known as forward pruning.

A widely used method of backward pruning in game trees is known as the alpha-beta procedure. The procedure functions as follows: Consider the diagram of Figure 4, which represents a game tree wherein the square nodes are the position at which it is Player A's turn to move, and

BREADTH-FIRST SEARCH

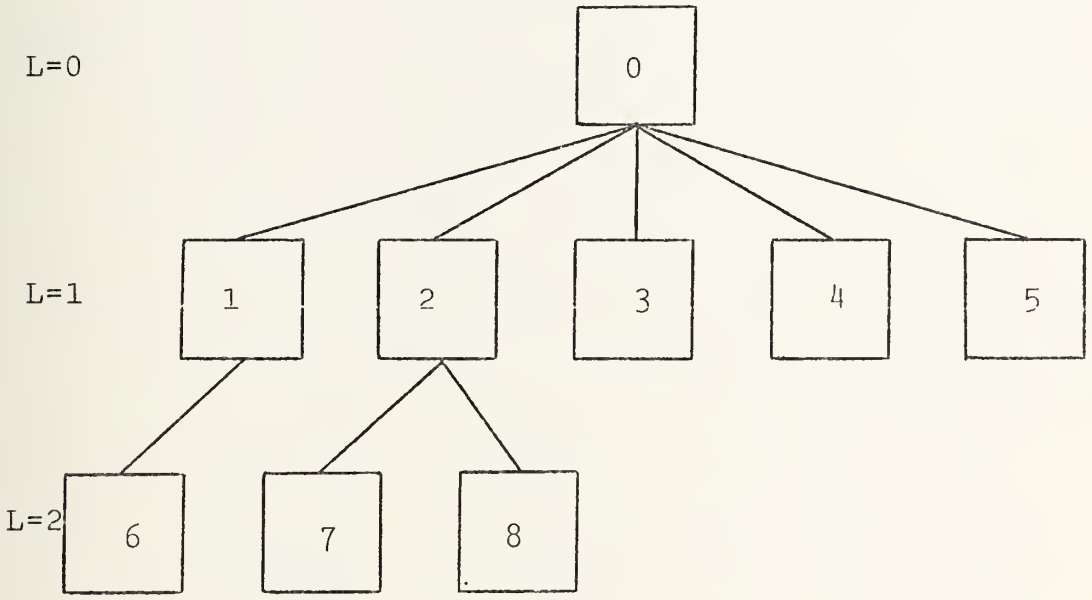


Figure 2.

DEPTH-FIRST SEARCH

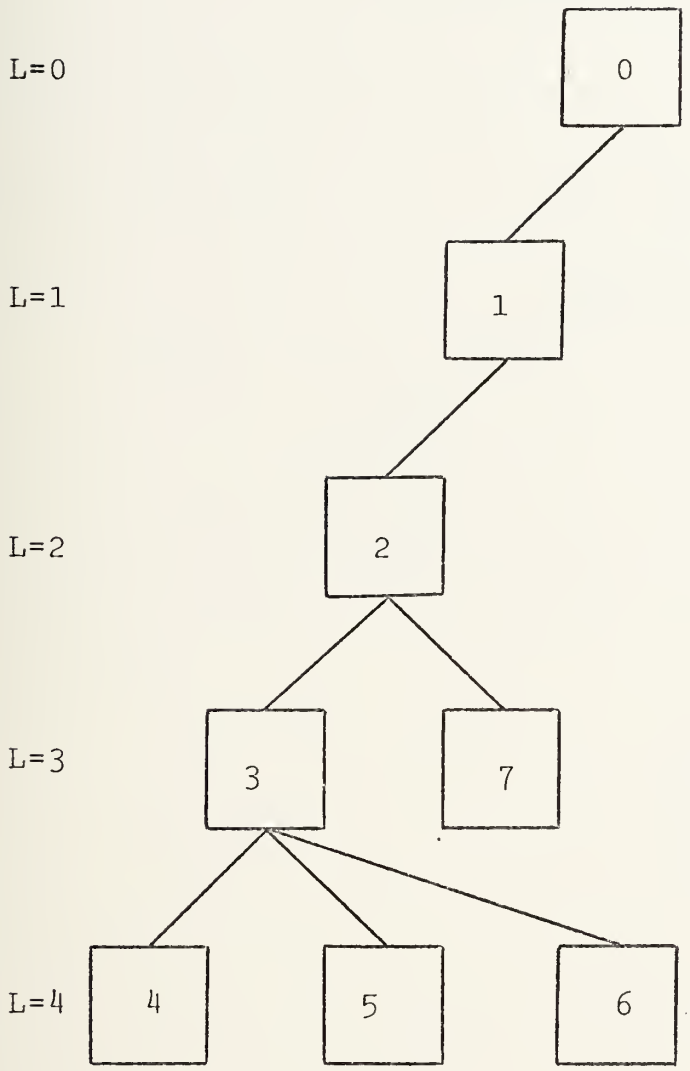


Figure 3.

ALPHA CUTOFF

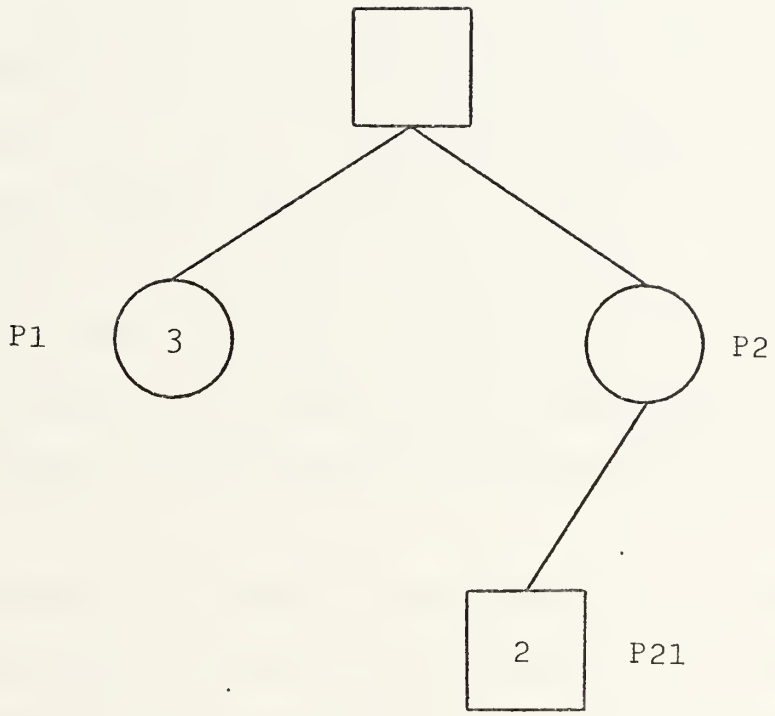


Figure 4.

the round nodes are the positions at which it is Player B's turn to move. Player A is the computer.

At the initial node, Player A has two options (branches) leading to nodes P1 and P2. The computer has obtained a (possibly backed-up) value for node P1 of V1=3. The alpha-beta procedure sets alpha=3 at P2. Now when node P21 is generated and evaluated, and the value V21 is found to be less than alpha, the procedure is said to have found an alpha cutoff. That is, the computer does not bother to generate nodes P22, P23,... and their successors, because it is already evident that node P2 is worse for Player A than node P1, since it is now known that Player B at node P2 has at least one choice which will result in a node of value less than that resulting from any possible branching from node P1. Therefore Player A should choose the branch to P1 as his next move.

Similarly, in Figure 5, the computer has assigned a (possibly backed-up) value of V11=4 for node P11. This value becomes the beta value for the nodes P121, P122, etc. Thus when the computer finds node P121 with a value of V21=8 greater than beta, it finds a beta cutoff. That is, it does not generate and evaluate node P131. Here again, it is assumed that Player B acting at node P1 will not choose the branch to node P12, because there is at least one successor to P12 which is greater in value than any of the successors of P11.

Note that this procedure depends on the accuracy and validity of the evaluations at each node. That is, if the

BETA CUTOFF

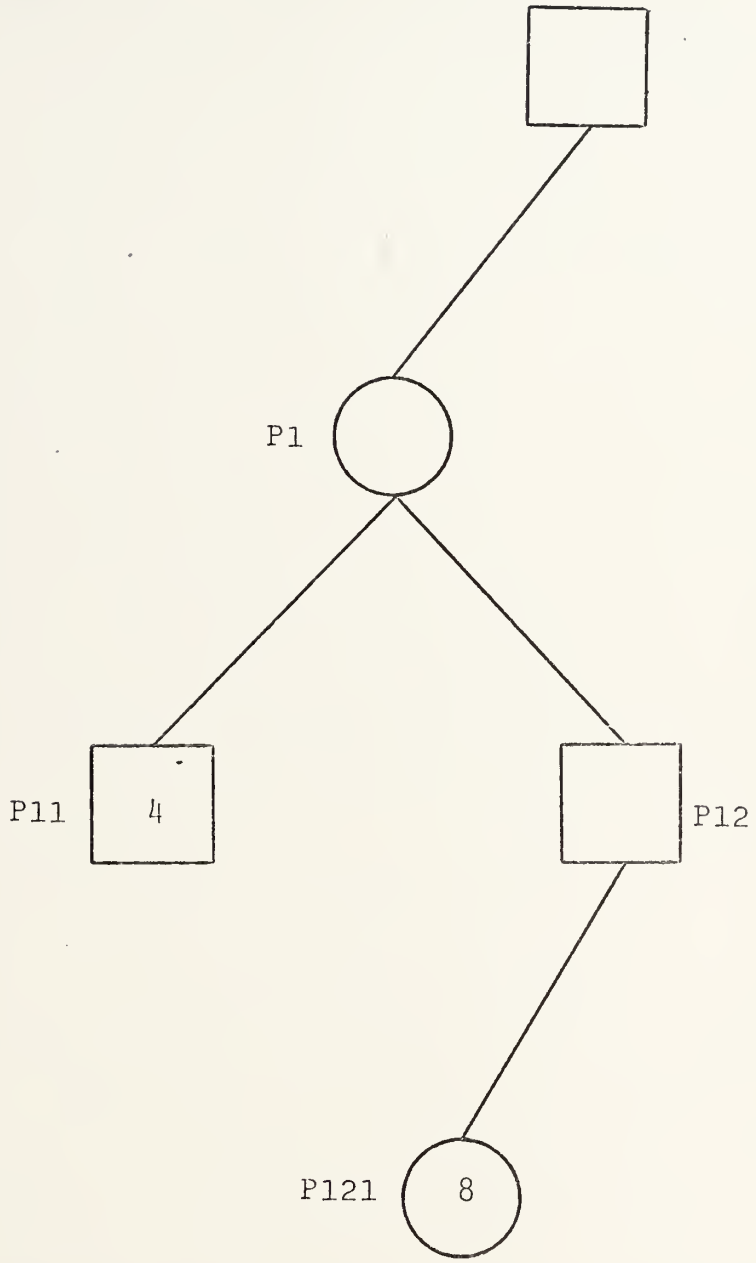


Figure 5.

heuristics used to generate the values are incorrect or incomplete, then the numbers assigned to each node become at best less meaningful, and the procedure may fail.

II. ADAPTIVE LOGIC

As described above, the success of the alpha-beta procedure - and indeed of any tree-search procedure - depends upon the validity of the evaluation function used to assign a value to a node. Thus one of the most difficult tasks facing the Artificial Intelligence programmer is the selection and generation of an evaluation function.

One method used to generate evaluation functions is to select a set of heuristics applicable to the problem, program the resulting operations and generate from the results of computation a set of parameters which describe the problem state. The values of the parameters - if they are correct and complete - can then be combined to form an evaluation of any node in the tree.

The set of the values of heuristically generated parameters can, of course, be combined in many ways. One method could be simply summing the values to obtain a total value for the node. Another method, more commonly used, is to form a weighted sum of the parameter values wherein the weights can be adjusted to obtain better results. A computer model of this technique is known as an adaptive logic element.

A. ADAPTIVE LOGIC ELEMENTS

Adaptive logic elements have been studied and analyzed for a number of years. These elements have found particular usefulness in the field of trainable pattern classifying

systems [7]. Training algorithms for these elements exist and are well-defined.

Figure 6 is a diagram of an adaptive linear threshold logic element, or Adaline [8]. The j^{th} inputs to the Adaline form a vector X_j, each element of which is multiplied by the corresponding element of a weight vector W_j. The output of the summer is then a scalar y_j with the value $y_j = X_j^T W_j$. Note that the first input x_{oj} is always +1; because of this the zeroth weight element w_{oj} controls the threshold of the Adaline. Thus the output y_j is biased by the value w_{oj}. The output sum y_j is now fed to the quantizer which produces the quantized output of the Adaline, q_j.

The adaptation machinery receives as inputs X_j, y_j, q_j and the desired response of the element d_j. If $q_j = d_j$, then the Adaline has performed properly and no action is taken (actually in some applications the element may be rewarded for correct response). If $q_j \neq d_j$, there exists an error e which is equal to $d_j - y_j$ or $d_j - X_j^T W_j$. In response to this error e the adaptation machinery causes the weights W_j to be changed. If this process is now repeated with the same input the Adaline will eventually arrive at the proper set of weights such that the desired output is obtained for the input X_j.

B. LEARNING WITH A TEACHER

In many of the fields for which adaptive logic elements have been used, for example pattern recognition, the desired output d_j for each input vector X_j is well known. In these fields, it is usually assumed that the set of patterns of

ADALINE

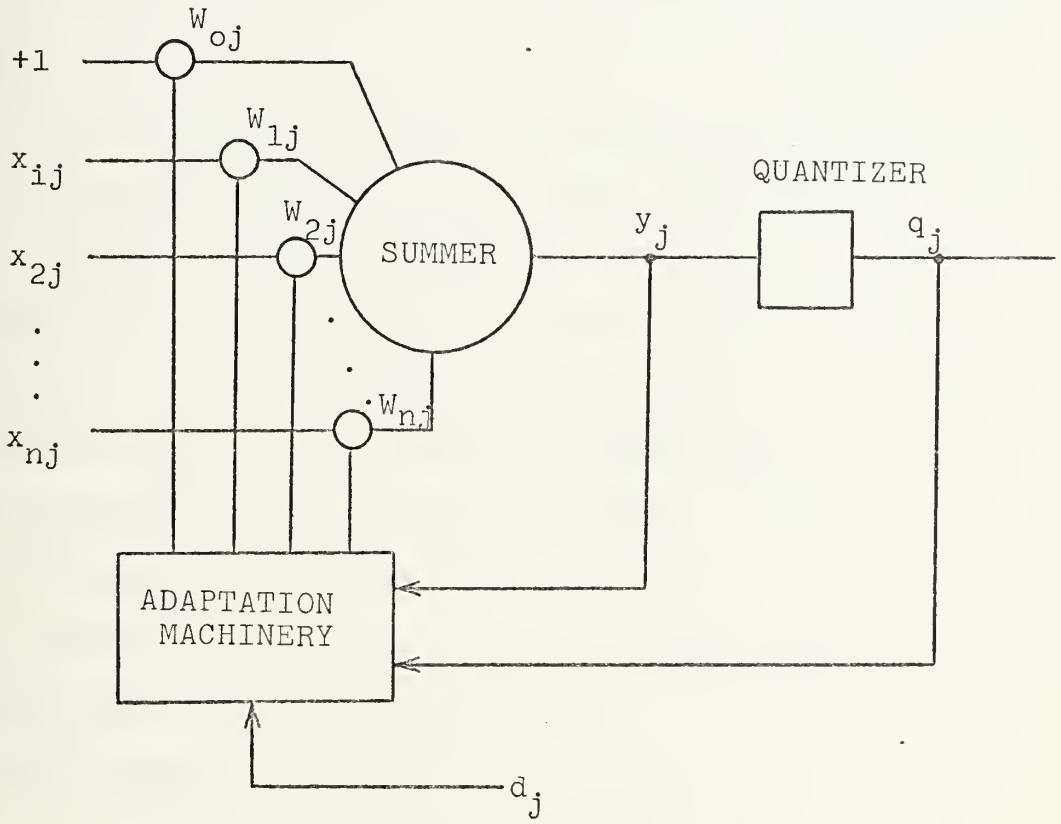


Figure 6.

interest is small compared to the total number of possible patterns, and that the set of patterns corresponds to a linearly separable function such that there is a set of weights which will provide the desired output for all input patterns of interest [9]. In cases where these assumptions are met, Adalines can provide high accuracy pattern recognition at very low cost for computer storage or processing time.

When an Adaline is used as described above, the adaptation machinery operates on the same input iteratively until the desired output is obtained, and after the element is trained for a large number of input patterns, the adaptation machinery is removed and the element will now be able to recognize most of the class of patterns for which it has been trained. This method of training is known as learning with a teacher. Adalines which learn with a teacher exhibit well-defined, exponential learning curves and will always be stable under known conditions [10].

The weight iteration rule

$$W_{j+1} = W_j + \frac{a}{n+1} e_j X_j$$

developed by Widrow and Hoff [16] ensures that $|e_j|$ will be reduced by the j^{th} adaptation, so long as

$$2 < a < 0.$$

C. LEARNING WITH A CRITIC

Adalines trained by learning with a teacher have proven to be a valuable tool in many fields. However, they cannot

be used when the desired output for each input pattern is unknown. To overcome this difficulty, and to expand the field of applications for Adalines, Widrow, Gupta and Maitra have proposed a new technique for training an Adaline, called learning with a critic [11].

In this method, also called selective-bootstrap learning, the output q_j of the Adaline is used to form the desired response signal d_j (see Figure 7). The bootstrap control signal b_j determines whether $d_j=q_j$ or $d_j=-q_j$. The control signal b_j is determined by an external evaluator, or critic. If the critic feels that the present decision (output q_j) is a member of a set of relatively successful decisions, then b_j is set such that $d_j=q_j$. In other words, the logic element is being told that the decision just made is a correct one. If, on the other hand, the critic feels that the decision was not a good one, then b_j is set such that $d_j=-q_j$.

In the first case, called positive bootstrap adaptation or learning by reward, the Adaline will tend to maintain the responses that already exist. In the second case, called negative bootstrap adaptation or learning by punishment, the Adaline will tend to change its existing output.

Continued iterative application of b_j as determined by the critic results in the training of the Adaline. Note that while learning with a teacher is well-defined and specific, learning with a critic is qualitative in nature.

SELECTIVE BOOTSTRAP ADAPTATION

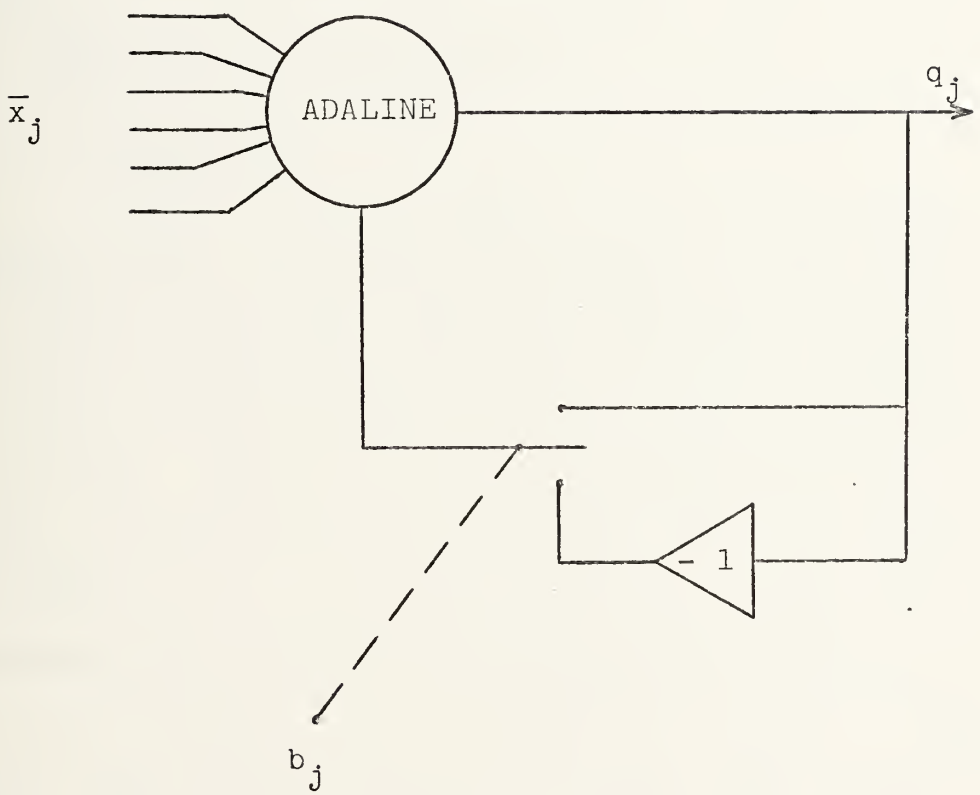


Figure 7.

III. THE PROBLEM OF CHESS

The attempt to program a computer to play grandmaster level chess is a problem which has fascinated the experts almost from the inception of the computer itself. Its solution has to date eluded the best efforts of the artificial intelligence community and the utilization of the most powerful computing machinery available.

A. DESCRIPTION OF THE PROBLEM

1. Why Chess?

Chess is merely a game. It is, to be sure, a very difficult game, and one which requires a great deal of skill in order to play well. But the qualities of chess which make it such an interesting problem for computer programming are threefold: Firstly, it is well-defined in that an exact model of the game can be programmed into the computer. Secondly, it is a very large problem; so large that no exhaustive search of all its possibilities could ever be carried out. Lastly, it is a member of a class of large problems which human beings can solve very well even though their resources seem to be even more limited than those of the computer. It is especially this third quality which makes chess such an interesting problem. It is generally felt that the ability to program a computer to play championship level chess will open new doors in all areas of artificial intelligence [12]. Indeed, it has been said that "If one

could devise a successful chess machine, one would seem to have penetrated to the core of human intellectual endeavor [13]."

2. Orders of Magnitude

Looking at a chessboard, it is difficult to imagine just how complex the game really is. There are only sixty-four squares on the board, and thirty-two pieces. Yet there are 10^{43} board positions possible and more than 10^{120} possible games. To put these numbers in perspective, consider that there are 10^{55} molecules comprising the entire earth [14]. Or consider a computer which could analyze one million board positions in one second; then it would take this computer 3.17×10^{29} years to analyze all board positions. After one million years of constant computation, the computer would have completed less than one thousand - billion - billionths of one percent of the total problem.

The magnitude of these numbers is really beyond the comprehension of men. Yet contrasted with these astronomical numbers are some small numbers associated with the human brain which are especially interesting. The brain, while having an enormous storage capacity, is inherently limited in many respects. First of all, it is very slow, operating in the range of hundredths of seconds [15]. Secondly, it has a very small short-term memory. In fact, it has been shown [16] that short-term memory can only store about seven "chunks" of information. ("Chunk," as distinct from "bit," has been chosen by G. A. Miller [16] to describe units of

information in the brain. A chunk is variable in size and can range from a single number to, in the case of a chess expert, a complete chessboard position.)

The question that arises from comparison of numbers such as 10^{43} with numbers such as 7 is this: How can a seemingly limited organism cope with such a vast problem? The answer to this question would probably solve the problem of computer chess - as well as many other problems - once and for all.

The key factor in the human's ability to play chess well is selection. As has been shown, a chess position presents the player with a choice of about thirty possible moves on the average. If he considers the opponent's possible responses to each move, there are now about 900 possibilities to be explored. Since it is known that a grandmaster often looks 5 or 6 moves ahead [17], if he considered every possibility he would consider on the order of 5×10^{14} moves! Yet it is known that the same grandmaster who looks 6 moves ahead only considers 50 - 100 possibilities at most [18]. Through some process not yet known, the skilled human player is able to drastically reduce the search space on which he operates to a level which he can handle.

If all the computer had to do was search these 100 nodes, and if the evaluation function used were reasonably time-efficient, then the competition between man and machine would be no contest. But the real problem for the computer is determining which hundred nodes to search.

The computer can partly make up for its lack of selectivity by brute force; that is, it can search many thousands of nodes while the human is considering his hundred. But this advantage is not enough to offset the human's advantage of selectivity.

Chess programming has therefore emphasized two major objectives: Increasing speed and power still more and increasing the computer's selectivity.

B. HISTORY OF COMPUTER CHESS

The first important publication on chess machinery was Shannon's paper in 1950 [19]. In his paper he described the basic approach to creating a chess program five years before the availability of computers capable of average play. In 1957 Newell, Shaw and Simon [20] predicted, conservatively, that a computer would be world champion within 10 years. The complexity of the game was greatly underestimated.

In 1967 Greenblatt [21] produced what is generally considered to be the most successful chess program. It plays at about the high amateur level.

Other notable chess programs which have been written include Northwestern Univeristy's Chess 3.0, 3.5 and 4.0 Kozdrowicki and Cooper's COKO II and III, Gillogly's Tech and Simon and Chase's MATER Program [22].

C. PROGRAMMING METHODS

Most chess programs, including Greenblatt's, Chess 4.0 and Samuel's highly successful checker player, use the well-known alpha-beta tree search procedure as their basis. They

also incorporate a plausibility generator as a forward pruning technique; that is, the first successor nodes to the problem situation node are ordered according to an evaluation function such that the most plausible or promising branches from the problem node are searched first.

A notable exception to the set of programs using alpha-beta is Tech [23] which, after an initial move ordering at the problem node, produces the search tree in a straightforward fashion by brute force as deep as time will allow. It is interesting to note that while the rating of Tech is only about 1300 - that of a weak amateur - in the second U. S. Computer Chess Championship Tech placed second, ahead of other programs which use sophisticated evaluation functions.

Kozdrowicki and Cooper's COKO III [24] uses a selective tree searching procedure whereby the entire tree is stored in memory and the machine makes a probabilistic estimate of the most effective node on the tree for further search.

Simon and Chase's MATER Program is an attempt to cause the computer to act exactly as a human chess player. From studies of protocols and human decision-making, they have constructed a program which contains all the verifiable elements of the human decision-making process. In imitating what is known of the structure of the brain, these researchers hope to be able to duplicate its function [25]. Although this approach has been likened to trying to debug a program without a computer [26], this study should bring greater knowledge concerning not only chess but the nature of all human problem-solving.

Botvinnik [27] believes that it is important to know which pieces are able to reach a certain square or sector of the board in a certain number of half-moves (plies). This establishes for each piece what Botvinnik calls a "horizon." Planning would include only those pieces whose horizon is sufficient to aid in attaining the goal. Some of Botvinnik's ideas have been programmed, but as yet no complete program incorporating all of his ideas has been announced.

IV. PLAYER AND PRO

PLAYER and PRO are two programs written for the purpose of attacking the chess problem. PRO is in essence the adaptation machinery of an adaptive logic element which is used in the training of PLAYER using published positions with known best moves. Actual games are played by PLAYER alone; no learning takes place during a game.

A. APPROACH TO THE PROBLEM

As has been shown, present day computer chess programs utilize the speed and power of modern computers to search thousands of nodes in order to arrive at a decision. This is not to say that their programmers did not use very sophisticated techniques in their designs in order to limit the search; however, the search carried out is still orders of magnitude greater than that carried out by a man. The exception is the MATER Program of Simon and Chase, which as described attempts to imitate human techniques.

The approach of PLAYER is different from either of these. PLAYER does not try to imitate a human, nor does it use brute force to search through a great many nodes. PLAYER was designed to study the chess problem and, insofar as possible, to provide an ordering of possible moves based on the initial node and its first successors only. A primary goal was to gather as much information as possible from the initial state.

PLAYER attempts to take advantage of two facts: The first is that a computer has perfect memory; that is, assuming it is functioning properly, it will not "miss" an opening or "not see" a threatening piece. The computer can easily keep track of a large number of pieces, positions and parameters as long as they are expressed in terms it can deal with. The second fact is a simple logical truth: All chess positions, assuming perfect play, terminate in either a win, a loss or a draw [28]. Therefore any chess position is implicitly either a winning position, a losing position or a draw position. If it could be assumed that a game is being played by two perfect chess players, a winning position could only result in a win, a losing position only a loss, and a draw position only a draw. For example, since White always has the advantage of the opening move [29], White should always win and Black should always lose. That this does not happen, even at the grandmaster level, is a testament to the fact that there are no perfect players.

The rub is, of course, that just as there are no perfect players, there is no one who can say for sure, for more than a very few given board situations, whether that situation is a winning, losing or draw position. It can be postulated that there is a set of parameters which can describe any board position in these terms; so far no one has found it.

The approach of PLAYER is to study the problem of parameter generation and evaluation by surveying the current board situation in terms of winning, losing or drawing, and to find a move which will result in another situation no worse than

the present situation. In order to do this PLAYER must evaluate all possible moves, but only to a depth of ½ ply. PLAYER therefore tries to gather as much information possible from a study of the original problem node and its first successors.

B. PLAYER

PLAYER is a computer program written entirely in FORTRAN. PLAYER consists of a short MAIN program and five subroutines which will be described below.

1. Data Structure

The chessboard is represented as an eight-by-eight array as shown in Figure 8. In all double-subscripted arrays, I represents the rank and J the file of the location specified. The following arrays form the basic data structure of PLAYER:

OCCSQ(8,8) contains, in the location where a piece rests, the value of that piece. White pieces are coded as plus and Black pieces as minus.

MPC(16) and OPC(16) contain the locations, in the form 10xI+J, of the machine and opponent pieces respectively. Each piece has a specific number assigned to it: the pawns 1 - 8, the knights 9 and 10, the bishops 11 and 12, the rooks 13 and 14, the queen 15 and the king 16.

MVALPC(16) and OVALPC(16) contain the values of each piece according to the standard P=10, N=30, B=32, R=50, Q=90 and K=9900. If a pawn is promoted, its value

CHESSBOARD REPRESENTATION

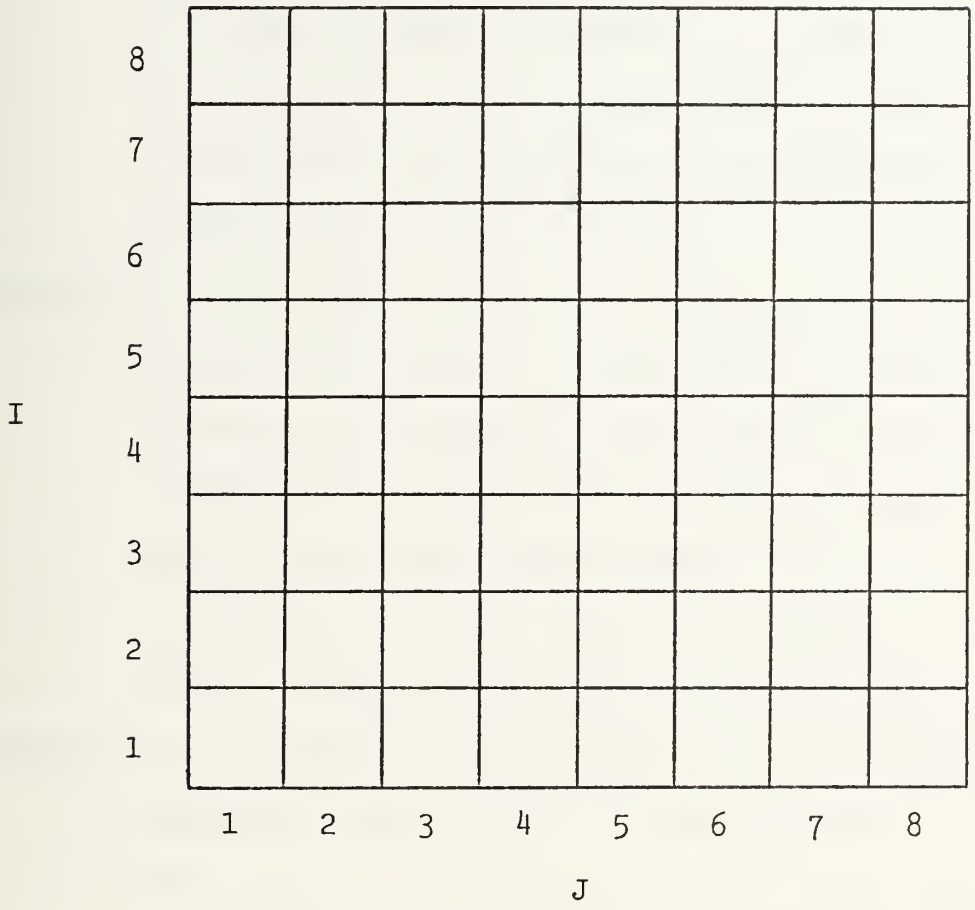


Figure 8.

is changed accordingly, usually to that of a queen. PLAYER can accept any number of pawn promotions to any piece value although for its own pawn promotions it always chooses to promote to a queen. (Lack of this facility cost COKO III a game [30]) If a piece is captured its value (and corresponding location in OCCSQ and MPC or OPC) is set to zero.

MPOSMV(N) and OPOSMV(N) contain the possible moves of the machine and opponent respectively. Moves are encoded as 100 times the piece number plus the square to which it can move. For example, a possible move of 242 means that piece number 2 can move to square 42 (or 4,2). Castle moves are specially coded as 98 for castle left or 99 for castle right.

MPHMV(N) and OPHMV(N) contain phantom moves for machine and opponent respectively. A phantom move is not a legal move, but represents a conceivable move under certain circumstances. For instance, assume that the computer's piece number 9 has a possible move to square 57, which is unoccupied. Now if square 57 subsequently becomes occupied by another machine piece, then the computer no longer has MPOSMV(N)=957, because it cannot capture its own piece. However there will now be generated MPHMV(M)=957, indicating that if the piece on square 57 is captured, then piece number 9 can recapture. Phantom moves are also used to indicate double attacks, as when a

rook rests behind another rook. The rear rook has phantom moves "through" the front rook to indicate that both can attack the same square. The same holds true for rooks and a queen, bishops and a queen, and pawns and a bishop or queen. Any number of pieces in line can be handled. All PHMV's (and POSMV's) are deleted if they place the friendly king in check (that is, if the piece is pinned).

MATTSQ(8,8) and OATTSQ(8,8) contain, for each square, the sum of value and sum of the number of pieces attacking that square, for machine and opponent respectively. Thus if a queen and pawn are both attacking the same square - say square 22 - then MATTSQ(2,2) would contain 10002 (100 x (90+10)+2). As in the case of MPHMV and OPHMV, MATTSQ and OATTSQ are filled to reflect the fact that a Bishop and a Queen, for example, on the same diagonal can both attack the same square. It is because of this feature that PLAYER can predict the results of exchanges without a tree search. However, in order to make its prediction, PLAYER assumes that each side will capture with its least valuable piece, given a choice. This assumption may not be valid if, for example, the opponent sacrifices a high-value piece for positional advantage.

MPAR(L) and OPAR(L) contain the values of the evaluation parameters for machine and opponent respectively.

At present there are 12 parameters developed for the opponent and 18 for the machine. These parameters will be discussed in greater detail below.

NWT(L) contains the set of weights used by PLAYER in conjunction with MPAR and OPAR to make a decision. NWT is the array that is modified by PRO in the training phase of PLAYER.

MCAS,OCAS,MCA,OCA,MENP,OENP are six important flags which are used throughout the program to indicate the status of castling and en passant capture possibilities.

2. MAIN

MAIN is a fairly short program which reads MPC, OPC, the six flags and the opponents move, calls SITU with this information and based on the output from SITU, calls SITU again for each possible move of the machine. MAIN saves the best move returned from SITU as determined by MPAR and OPAR. It then prints the move chosen as well as the new values for MPC, OPC and the six flags.

3. SITU

SITU is the controlling subroutine for the other four subroutines in the program. It calls the other subroutines in a prescribed order, but can be accessed as though it were the machine's or the opponent's turn to move.

SITU first calls EXEC to execute the move just made. The move made may be an actual move from the opponent or a possible move by the computer. Now SITU calls MOVES, CHECK and PARGEN to determine the set of parameters for the player

whose turn it is to move. The sets of parameters MPAR and OPAR are returned to MAIN.

4. EXEC

EXEC is the subroutine which executes a move. The move executed may be an actual move, phantom move or possible move, depending on MAIN's call to SITU. In addition to normal moves, EXEC also handles castling, en passant capture and pawn promotion. It also changes the status of the six flags if a castling move is made or if a pawn moves so as to be in a position to be captured en passant. (Castle flag changes are also made if either rook moves from its original position, if the king moves or if any of the squares over which the king would pass while castling are such that the king would be in check, or, of course, if the king is in check.) If a pawn is promoted, EXEC changes the value in MVALPC or OVALPC and OCCSQ of the piece concerned to that of a queen. If a piece is captured by the move executed, EXEC changes the pertinent values of MPC and MVALPC (or OPC and OVALPC) and OCCSQ to zero.

5. MOVES

MOVES is the subroutine which develops all possible moves for each player based on a given situation. It is also used, though not concurrently, to determine whether a player is in check.

If MOVES is called with NCHK=0, it will return the set of all possible moves, all phantom moves, and attacking squares for one player. The set of moves returned includes special moves such as en passant capture and castling.

If MOVES is called with NCHK=1, it will return the value NCHK=0 if the player is not in check and the value NCHK=1 if the player is in check. NCHK is the only meaningful parameter returned when MOVES is called with NCHK=1.

6. CHECK

One of the basic concepts of chess is that of a pinned piece. A piece is pinned if its movement will result in its own king being in check; any such move is illegal. Since MOVES will not detect a pinned piece, subroutine CHECK was written to accomplish this detection.

CHECK receives as input the possible moves generated by MOVES, saves all necessary arrays and flags, calls EXEC to execute each possible move and then calls MOVES with NCHK=1 to determine if the possible move is legal. Illegal moves are eliminated from the list of possible moves. The process is repeated for each phantom move, and in all cases MATTSQ (or OATTSQ) is reduced accordingly (except in the case where the MATTSQ element is the location of the opposing king, since a pinned piece can still check).

The combination of MOVES and CHECK ensures that only legal moves and legitimate attacking squares are presented to PARGEN for evaluation.

7. PARGEN

PARGEN receives as input from EXEC, MOVES and CHECK the arrays OCCSQ, MPC, OPC, MPOSMV, MPHMV and MATTSQ (or OPOSMV, OPHMV and OATTSQ) and uses them to generate a set of parameters which can be used to evaluate the current board

position. Note that the current board situation presented to PARGEN can be either the original position or that arising from the execution of one of the machine's possible moves.

Parameters 1 through 11 are generated for both players; parameters 12 through 17 are generated for the machine only; parameter 20 is the weighted sum of all the other parameters.

The following is a description of the parameters generated by PARGEN:

PAR(1) Mobility: PAR(1) is the number of all possible legal moves from the current board position. If PAR(1)=0, then the position is checkmate.

PAR(2) Material Value: PAR(2) is the sum of the values of all the pieces still on the board, exclusive of the king (the king cannot be lost).

PAR(3) Pawn advancement: PAR(3) is the sum of the squares of the ranks of each pawn. This parameter rewards pawn advancement in the hope of eventual pawn promotion.

PAR(4) Center Control: PAR(4) gives credit for each pawn in the center four squares. It also gives credit for attacking opposing center pawns or defending own center pawns. The importance of center control in chess has been known for centuries.

PAR(5) Pressure: PAR(5) gives points for reducing the number of opponent moves to one. It also gives points if the opponent can make only king moves.

PAR(6) King Protection: PAR(6) gives points for own pieces directly in front of own king. This parameter was designed

to reward good castle-pawn structure.

PAR(7) Castle: PAR(7) gives 100 points if any possible move is a castle move. It gives 200 points if the player has castled.

PAR(8) Early Development: PAR(8) gives points for moving own pieces (bishop, knight or queen) off the back rank.

PAR(9) Attack I: PAR(9) gives points for a queen or rook attacking the opponent's back rank. It also gives points for a queen or bishop attacking the opponent king's castle pawn structure.

PAR(10) Attack II: PAR(10) gives points for attacking any opponent pieces.

PAR(11) Pawn Development: PAR(11) deducts points if a pawn is blocked by an opposing pawn. This parameter is designed to operate in conjunction with PAR(3).

PAR(12) Exchange: PAR(12) determines the result of all possible exchanges. It deducts points severely if the exchange will be lost; it gives points if the exchange would be won (but note that the opponent will probably make another move rather than participate in a losing exchange).

PAR(13) Pawn Pressure: PAR(13) deducts points if an opposing pawn can threaten a piece in one move without being captured. This is an attempt to eliminate fruitless moves.

PAR(14) Danger I: PAR(14) deducts points if the opponent can move a queen or rook to an open file without being captured. It deducts further points if opponent will have more pieces attacking the back rank than the machine has defenders.

PAR(15) Danger II: PAR(15) deducts points if the opponent can move a queen or bishop to an open diagonal or if he can move a queen or rook to a position such that, without being captured, he can attack the machine's castle-pawn structure. It deducts further points if the attackers of the machine's pawn structure outnumber the defenders.

PAR(16) Attack III: PAR(16) gives points for the machine being in position to move to an open file in the same manner as PAR(14).

PAR(17) Attack IV: PAR(17) gives points for the machine being in position to move to an open diagonal or to attack the opponent's castle-pawn structure in the same manner as PAR(15).

After determining these parameters, PARGEN multiplies the resultant parameter vector by the input weight vector NWT, and transfers the result via SITU to MAIN.

C. PRO

PRO is a short program which carries out the adaptation algorithm of Widrow and Hoff [31]:

$$W_{j+1} = W_j + \frac{a}{n+1} e_j X_j .$$

PRO reads the value of the weights and parameters as provided by PLAYER, performs the adaptation algorithm, and prints the results.

The critic function described in the section on adaptive logic is performed by the programmer. The programmer determines the worth of a PLAYER decision by consulting standard texts, asking PLAYER to decide a move from a published position,

and comparing PLAYER's move to that of the expert involved in the published position. Any arbitrary position can be presented to PLAYER for training or for testing.

D. SUMMARY

As has been shown, the basic working method of most other chess programs is as follows: First successor nodes to the problem situation are generated and ordered in a plausibility hierarchy according to a heuristic value function. the N "best" nodes are expanded and searched as deeply as possible within real-time constraints, the search being guided by chess heuristics. The end nodes of the tree generated are evaluated, using the same or another set of heuristics, and the evaluation is "backed-up" to the problem node in order to decide which move to make. The purpose of searching through so many nodes is to attempt to make up for the fact that the original plausibility ordering is imperfect.

PLAYER and PRO are based on the postulate that there is a perfect value function in chess, that this value function depends on specific knowledge of every aspect of the given problem situation, and that this value function can be determined with the aid of adaptive logic.

The method of PLAYER and PRO is to generate the first successor nodes to the problem situation only, to evaluate these nodes as completely and specifically as possible, and to train the value function generator - the decision-maker - to make the best decision using adaptive logic techniques.

V. CONCLUSIONS

A. EVALUATION METHOD

The method chosen to evaluate the capabilities of PLAYER in ordering moves is one in which quantitative results are produced. It is felt that these results are more meaningful than a subjective appraisal based on actual play.

In order to test PLAYER, 40 chess positions were chosen from three chess texts. Most of the positions were taken from those appearing in actual games between grandmasters; others were positions used for instructional purposes. In each case it was assumed that the move made by the expert is in fact the best move. PLAYER assessed each position and printed all moves possible from that position, ordered according to its evaluation function. The rank assigned the recommended move by PLAYER was used as an indication of the worth of PLAYER's evaluation function for that position. In other words, if PLAYER were perfect, then it would always rank the recommended move as number one - it would always make the recommended move. The closer PLAYER is to perfection, the higher it will rank all recommended moves.

All positions evaluated are illustrated in the diagrams in Appendix A. In these diagrams, abbreviations for the chess pieces are used as follows: K=King, Q=Queen, R=Rook, B=Bishop, N=Knight and P=Pawn. Pawns are named according to their file (see Figure 9); that is, the pawn in the

CHESS NOTATION

BLACK

QR8	QN8	QB8	Q8	K8	KB8	KN8	KR8
QR7	QN7	QB7	Q7	K7	KB7	KN7	KR7
QR6	QN6	QB6	Q6	K6	KB6	KN6	KR6
QR5	QN5	QB5	Q5	K5	KB5	KN5	KR5
QR4	QN4	QB4	Q4	K4	KB4	KN4	KR4
QR3	QN3	QB3	Q3	K3	KB3	KN3	KR3
QR2	QN2	QB2	Q2	K2	KB2	KN2	KR2
QR1	QN1	QB1	Q1	K1	KB1	KN1	KR1

WHITE

Figure 9.

Queen's Rook (QR) file is called the Queen's Rook's Pawn, abbreviated QRP. A minus sign is used to indicate Black pieces; a plus sign is used for White. The moves are listed in standard chess notation, for example Q-QB3 means the Queen is moved to square QB3 (see Figure 9). Where no confusion can arise, the first letter of the piece designation or square designation may be omitted, as in Q-B3 vice Q-QB3 or P-K4 vice KBP-K4. An 'x' indicates capture, for example Queen takes Knight would be abbreviated QxN.

Where moves are listed in a line of text, as in Figure 20, for example, they are always listed in the order White move, Black move. Moves followed by an exclamation point (!) are judged - by the expert - to be very good moves; moves followed by two exclamation points (!!) are judged unusually good moves. The computer is playing White in all moves except moves number 21 to 23. Recommended moves are all underlined in the diagrams of Appendix A.

It should be emphasized that this test is a very demanding one. The forty situations include samples from the opening, middle and end games. There are piece sacrifices, developmental moves and positional moves included. Almost half the recommended moves in these selected situations were judged very good or unusually good by the standards of chess experts.

B. RESULTS

The results of the test of PLAYER are shown in Table I. The columns of this table show the number of the position

TABLE I.

Position Number	Comments	Total Moves	Ranking
1.	(!!)(S)	55	48
2.		39	7
3.		45	9
4.		47	15
5.		48	3
6.	(S)(!)	53	42
7.	(!)	45	21
8.	(!)	48	4
9.		45	6
10.		38	1
11.		37	10
12.		44	29
13.		41	7
14.	(!)	36	7
15.	(!)	42	1
16.	(!)	46	22
17.	(!!)(S)	50	31
18.		40	2
19.		20	1
20.		29	3
21.		20	4
22.		20	1
23.		27	1
24.	(!)	37	2

25.	(!)	38	4
26.		37	1
27.	(!)	35	9
28.	(!)	30	1
29.	(!)	22	1
30.	(!)	51	1
31.	(!!)	45	5
32.	(!)	39	11
33.	(!)	37	29
34.		37	1
35.		43	2
36.		41	1
37.		44	1
38.		42	2
39.		43	1
40		31	1

TABLE I. Continued.

considered, comments concerning the recommended move, the total number of moves evaluated by PLAYER for each position and PLAYER's ranking of the recommended move for each position.

The average number of moves per position generated by PLAYER for evaluation was 39. Of the 40 positions considered, PLAYER ranked the recommended move in the top ten moves 31 times. The recommended move was ranked in the top five moves 24 times, and 14 times PLAYER chose the recommended move exactly.

Piece sacrifice moves - designated by (S) in Table I - were all ranked very low; this is to be expected as PLAYER was programmed to avoid sacrifice if at all possible.

With two exceptions, all the moves ranked lower than ten were either sacrifice moves of 'very good' moves - or both. These are precisely the kinds of moves which require the most profound understanding of the subtleties of chess; that is, these are the moves which separate the master from the amateur.

PLAYER ranked one 'unusually good' move in the top five, ranked 9 'very good' moves in the top ten (7 of these in the top five) and found 4 'very good' moves exactly.

Judged in the context of other existing chess programs, PLAYER's performance is impressive. The results are sufficiently encouraging to warrant further investigation into what is demonstrably a viable technique.

Improvement in PLAYER's performance can be obtained by the generation and refinement of more parameters and by more

training in conjunction with PRO. In particular, a means should be found to determine when a piece sacrifice will result in a winning position or combination. In order to make a decision of this kind, the computer would have to be programmed to develop and examine a small set of plans. That is, the computer would select a few key pieces, and for each selected piece examine a few sacrifice moves to determine whether the projected sacrifice would result in positional gain. (The set of all possible sacrifices could not be examined because of time constraints.) If a means could be found to have the computer select the 'best set' of pieces and moves for possible sacrifice, the resulting program would for the first time approach the level of sophistication of the human mind.

APPENDIX A: EVALUATION POSITIONS

Position 1

-R		-B			-R	-K	
-P	-P	-Q			-P	-P	-P
		-P				-N	
						+N	
+P		+B	+P		-N		
	+Q						
	+P		+N		+P	+P	+P
		+R		+R		+K	

Recommended Move [32]

1. R-K8!!

Figure 10.

Positions 2 - 6

-R		-B			-R	-K	
-P	-P	-B		-Q	-P	-P	-P
		-N			-N		
		-P		-P			
				+P			
		+P	+B	+N	+N		+P
+P	+P				+P	+P	
+R		+B	+Q	+R		+K	

Recommended Moves [33]

2. N-Q5 Q-Q3,

then 3. B-KN5 N-K1,

then 4. Q-B2 B-Q1,

then 5. B-K3 P-Q3,

then 6. B-QN5!

Figure 11.

Positions 7 - 10

		-B		-R	-N	-K	
-R				-B	-P		
	-Q	-P		-P	-P		-P
+N	-P	+N		+P			-N
	+P		+P				
		+Q					+P
					+P	+P	+B
	+B		+R		+R	+K	

Recommended Moves [34]

7. B-K4! R-B2,

then 8. Q-B3! B-Q2,

then 9. N×B N×N,

then 10. N×P.

Figure 12.

Positions 11 - 12

			-R			-K	
					-P	-P	-P
-P	-P						
			-Q	+P			
+P		-R	+P		+R		
						+P	
	+Q						+P
+R						+K	

Recommended Moves [35]:

11. Q×P R×QP,

then 12. R×R.

Figure 13.

Position 13

		-R				-K	
		-R		-N	-P	-B	-P
						-P	
-P					-Q		
+P	-P		-P				
	+P		+Q				
		+P	+B		+P	+P	+P
		+R	+N	+R		+K	

Recommended Move [36]

13. R-K4.

Figure 14.

Positions 14 - 16

-R		-B			-R	-K	
				-Q		-P	-P
				-P			
-P	-P		-P				
			+P				
			+Q	+P		+B	
+P					+P	+P	+P
+R					+R	+K	

Recommended Moves [37]

14. KR-B1! B-R3,

then 15. R-B7! Q-K1,

then 16. Q-R3!

Figure 15.

Positions 17 - 18

-R				-R			
	-B				+Q	-P	-K
-P	-Q			-P	-B		
	-P	-N		+N			
		+N		+P			
+P	+P				+P	+P	+P
		+R	+R			+K	

Recommended Moves [38]

17. N-Q7!! N×N,

then 18. R×N.

Figure 16.

Positions 19 - 20

-R	-N	-B	-Q	-K	-B	-N	-R
-P	-P	-P	-P	-P	-P	-P	-P
+P	+P	+P	+P	+P	+P	+P	+P
+R	+N	+B	+Q	+K	+B	+N	+R

Recommended Moves [39]

19. P-K4 P-K4,

then 20. N-KB3.

Figure 17.

Position 21

-R	-N	-B	-Q	-K	-B	-N	-R
-P	-P	-P	-P	-P	-P	-P	-P
				+P			
+P	+P	+P	+P		+P	+P	+P
+R	+N	+B	+Q	+K	+B	+N	+R

Recommended Move [40]

21. ... P-K4.

Figure 18.

Positions 22 - 23

-R	-N	-B	-Q	-K	-B	-N	-R
-P	-P	-P	-P	-P	-P	-P	-P
			+P				
+P	+P	+P		+P	+P	+P	+P
+R	+N	+B	+Q	+K	+B	+N	+R

Recommended Moves [41]

22. .... P-Q4,

then 23. P-QB4 P-K3.

Figure 19.

Positions 24 - 26

			-R				
	-P			-B	-P		-K
	+P		-Q			-P	-P
	+R	-P	+P	-P			-N
				+P			
				+Q	+N		+N
		+B			+P	+P	
						+K	

Recommended Moves [42]

24. Q-B3! P-B3,

then 25. N-Q2! N-B5,

then 26. K-B1

Figure 20.

Positions 27 - 29

				-N	-R		-K
	-P		-B	-R			-P
-P		-P		-P	+N	-P	+N
		+P	-P	+P		+P	
+P	+P		+P				
							+R
							+P
					+R	+K	

Recommended Moves [43]

27. R/R3-KB3! N×N,

then 28. R×N! R×R,

then 29. KP×R!

Figure 21.

Position 30

-R		-R					
-P	-P		-Q	-N	-K		-P
				+N	-P		
			-P				
						+Q	
+P	+P				+P	+P	+P
		+R		+R		+K	

Recommended Move [44]

30. N-N5ch!

Figure 22.

Position 31

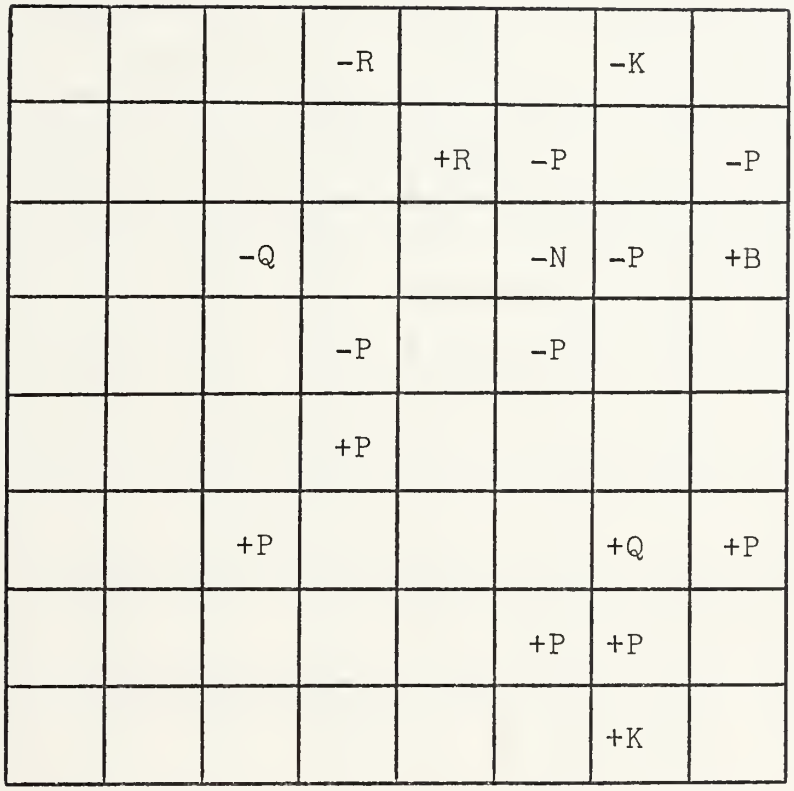
	-K					-R	-R
-P	-B						-N
	-Q	-P	-P			+Q	+P
	-P	-B		-P			
				+P			
	+P	+N	+P			+N	
	+P	+P	+B				
		+K			+R		+R

Recommended Move [45]

31. Q-N7!!

Figure 23.

Positions 32 - 34



Recommended Moves [46]

32. Q-K5! R-R1,

then 33. R-B7! Q-K3,

then 34. Q×Q

Figure 24.

Positions 35 - 37

	-K	-R		-B			-R
-P	-P	-P	+Q	+R			
				+N	-P	-N	-Q
+P		+P				-P	-P
		+P					
					+P	+P	+P
			+R		+N	+K	

Recommended Moves [47]

35. N-QB6 P×N,

then 36. R-QN1 K-R1,

then 37. Q-B1 mate.

Figure 25.

Positions 38 - 40

	-R					-K	
-Q					-P	-P	
-B				-P			-P
			-P				
-P	-R		+P				+Q
			+B				
+P	+P	+R			+P	+P	+P
		+R				+K	

Recommended Moves [48]

38. R-B8 R×R,

then 39. R-B8 Q×R,

then 40. Q-Q8 mate.

Figure 26.

```
      IMPLICIT INTEGER(O)
      DIMENSION MPC(16),OPC(16),MVALPC(16),OVALPC(16),
     . OCCSQ(8,8),MPOSMV(75),OPOSMV(75),MPAR(20),OPAR(20),
     . JOCCSQ(8,8),OS(65),MTHR(20)
      DATA MVALPC/8*10,2*30,2*32,2*50,90,9900/,
     . OVALPC/8*10,2*30,2*32,2*50,90,9900/
      DO 15 I=1,8
      DO 15 J=1,8
      OCCSQ(I,J)=0
   15 CONTINUE
      READ(5,150) (MPC(I),I=1,16)
  150 FORMAT(16I4)
      READ(5,150) (OPC(I),I=1,16)
      READ(5,200) MENP,OENP,MCAS,OCAS,MCA,OCA
  200 FORMAT(6I4)
      READ(5,210) OMV
  210 FORMAT(I8)
      WRITE(6,215) OMV
  215 FORMAT(3X,I8)
      DO 300 K=1,16
      IF(MPC(K).EQ.0) GO TO 250
      I=MPC(K)/10
      J=MPC(K)-10*I
      OCCSQ(I,J)=MVALPC(K)
      GO TO 260
  250 MVALPC(K)=0
  260 IF(OPC(K).EQ.0) GO TO 280
      I=OPC(K)/10
      J=OPC(K)-10*I
      OCCSQ(I,J)=-OVALPC(K)
      GO TO 300
  280 OVALPC(K)=0
  300 CONTINUE
      MTURN=1
      CALL SITU(MPC,OPC,OCCSQ,MVALPC,CVALPC,MPOSMV,MNT,
     . MTURN,MCA,OCA,MPAR,OPAR,OMV,MCAS,OCAS,MENP,OENP)
      DO 1000 K=1,20
      MTHR(K)=MPAR(K)-OPAR(K)
 1000 CONTINUE
      WRITE(6,2400)
 2400 FORMAT(12X,'MCAS',5X,'OCAS',5X,'MENP',5X,'OENP',
     . 5X,'MCA',5X,'OCA'//)
      WRITE(6,1800) MCAS,OCAS,MENP,OENP,MCA,OCA
 1800 FORMAT(6X,6I9//)
      IF(MNT.EQ.0) GO TO 4410
      I27=0
 1827 MGTR=-10000
      MATE=0
      DO 4000 N=1,MNT
      IF(I27.EQ.0) GO TO 2005
      IF(N.NE.MRMV) GO TO 4000
 2005 CONTINUE
      DO 2150 K=1,16
      K1=K+16
      K2=K+32
      K3=K+48
      OS(K)=MPC(K)
      OS(K1)=OPC(K)
      OS(K2)=MVALPC(K)
      OS(K3)=CVALPC(K)
```

```
2150 CONTINUE
     DO 2250 I=1,8
     DO 2250 J=1,8
     JOCCSQ(I,J)=OCCSQ(I,J)
2250 CONTINUE
     M1=MCAS
     M2=OCAS
     M3=MCA
     M4=OCA
     M5=MENP
     M6=OENP
     MP=MPOSMV(N)/100
     MMV=100*MPOSMV(N)+MPC(MP)
     IF(MPOSMV(N).LT.100) MMV=MPOSMV(N)
     MTURN=-1
     CALL SITU(OPC,MPC,OCCSQ,OVALPC,MVALPC,OPOSMV,ONT,
    . MTURN,OCA,MCA,OPAR,MPAR,MMV,OCAS,MCAS,OENP,MENP)
     DO 2000 K=1,20
     MPAR(K)=MPAR(K)-OPAR(K)
2000 CONTINUE
     IF(OPAR(20).EQ.0) MPAR(20)=MPAR(20)+3000
     IF(OPAR(20).EQ.0) MATE=1
     DO 2200 K=1,20
     OPAR(K)=MPAR(K)-MTHR(K)
2200 CONTINUE
     MRK=MPOSMV(N)/100
     IF(MRK.NE.16) GO TO 2348
     IF(M3.EQ.100) GO TO 2348
     MPAR(20)=MPAR(20)-50
2348 WRITE(6,2710) MPOSMV(N),(MPAR(K),K=1,17),MPAR(20)
2710 FORMAT(2X,I5,2X,17I5,2X,I6)
     IF(MPAR(20).GT.MGTR) MPMV=N
     IF(MPAR(20).GT.MGTR) MGTR=MPAR(20)
     IF(I27.EQ.0) GO TO 3129
     IF(MPAR(5).GT.0) WRITE(6,2863)
2863 FORMAT(5X,'CHECK')
     WRITE(6,2853) (MPC(K),K=1,16)
2853 FORMAT(1614)
     WRITE(6,2853) (OPC(K),K=1,16)
     WRITE(6,2859)
2859 FORMAT(2X,'MENP',2X,'OENP',2X,'MCAS',2X,'OCAS',
    . 2X,'MCA',2X,'OCA')
     WRITE(6,2856) MENP,OENP,MCAS,OCAS,MCA,OCA
2856 FORMAT(6I6)
     GO TO 4450
3129 DO 3150 K=1,16
     K1=K+16
     K2=K+32
     K3=K+48
     MPC(K)=CS(K)
     OPC(K)=OS(K1)
     MVALPC(K)=OS(K2)
     OVALPC(K)=OS(K3)
3150 CONTINUE
     DO 3250 I=1,8
     DO 3250 J=1,8
     OCCSQ(I,J)=JOCCSQ(I,J)
3250 CONTINUE
     MCAS=M1
     OCAS=M2
     MCA=M3
     OCA=M4
     MENP=M5
     OENP=M6
4000 CONTINUE
     MLS=MPOSMV(MPMV)/100
     IF(MLS.EQ.0) MLS=1
     MLS=MPC(MLS)
     WRITE(6,4255) MPOSMV(MPMV),MLS
4255 FORMAT(5X,'MY MOVE IS',I8,I4)
     IF(MATE.NE.0) GO TO 4300
     GO TO 4450
```

```
4300 WRITE(6,4315)
4315 FORMAT(5X,'CHECKMATE')
     GO TO 4450
4410 WRITE(6,4425)
4425 FORMAT(5X,'I RESIGN')
4450 IF(I27.EQ.1) GO TO 4510
     I27=1
     GO TO 1827
4510 STOP
     END
     SUBROUTINE SITU(MPC,OPC,OCCSQ,MVALPC,OVALPC,MPOSMV,
    . MNT,MTURN,MCA,OCA,MPAR,OPAR,JMV,MCAS,OCAS,MENP,OENP)
     IMPLICIT INTEGER (O)
     DIMENSION MPC(16),OPC(16),OCCSQ(8,8),MVALPC(16),
    . OVALPC(16),MPAR(20),OPAR(20),MPHMV(75),OPHMV(75),
    . MATTSQ(8,8),OATTSQ(8,8),MPOSMV(75),OPOSMV(75)
     DO 500 I=1,8
     DO 500 J=1,8
     OCCSQ(I,J)=OCCSQ(I,J)*MTURN
 500 CONTINUE
     NTURN=-1
     CALL EXEC(JMV,MENP,OENP,OCAS,OCA,OCCSQ,MPC,OPC,
    . OVALPC,NTURN)
     DO 2129 K=1,16
     IF(MPC(K).EQ.0) MVALPC(K)=0
2129 CONTINUE
     OCAS=0
     IF(OCA.GE.50) OCAS=2
     IF(OCA.EQ.10) OCAS=1
     IF(OCA.EQ.-10) OCAS=-1
     MCAS=0
     IF(MCA.GE.50) MCAS=2
     IF(MCA.EQ.10) MCAS=1
     IF(MCA.EQ.-10) MCAS=-1
     NCHK=1
     ONPHT=0
     IF(MTURN.EQ.-1) ONPHT=1
     NTURN=-1
     CALL MOVES(MPC,OPC,OCCSQ,OVALPC,OATTSQ,OPOSMV,OPHMV,
    . ONPHT,MENP,OCAS,OCA,NTURN,NCHK,ONT)
     IF(NCHK.EQ.1) MCAS=2
     ONPHT=0
     IF(MTURN.EQ.-1) ONPHT=1
     NTURN=-1
     CALL MOVES(MPC,OPC,OCCSQ,OVALPC,OATTSQ,OPOSMV,OPHMV,
    . ONPHT,MENP,OCAS,OCA,NTURN,0,ONT)
     IF(ONT.EQ.0) GO TO 1725
     NTURN=-1
     MNPHT=0
     IF(MTURN.EQ.-1) MNPHT=1
     CALL CHECK(OPC,MPC,OCCSQ,MVALPC,MATTSQ,MPOSMV,MPHMV,
    . MNPHT,MENP,MCAS,MCA,NTURN,MNT,OPOSMV,OPHMV,ONPHT,
    . OVALPC,OATTSQ,ONT,OENP,OCAS,OCA)
     IF(MCAS.EQ.2) GO TO 1725
     ML=0
     IKA=MPC(16)/10
     DO 1705 JKA=2,4
     IF(OATTSQ(IKA,JKA).NE.0) ML=1
1705 CONTINUE
     MR=0
     DO 1710 JKA=6,7
     IF(OATTSQ(IKA,JKA).NE.0) MR=1
1710 CONTINUE
     IF(ML.EQ.0) GO TO 1711
     MCAS=1
     IF(MCA.EQ.-10) MCAS=2
     IF(MR.NE.0) MCAS=2
     GO TO 1725
1711 MCAS=0
     IF(MCA.EQ.-10) MCAS=-1
     IF(MR.NE.0) MCAS=-1
1725 CONTINUE
```

```
      NTURN=-1
      OPAR(1)=0
      IF(MTURN.EQ.-1) OPAR(1)=1
 1800 CALL PARGEN(OPOSMV,ONT,OATTSQ,OPAR,OVALPC,OCCSQ,
     . OCA,OPC,MPC,NTURN,OPHMV,ONPHT)
      IF(MTURN.NE.1) GO TO 1825
      DO 1810 I=1,8
      L=9-I
      WRITE(6,1815) (OCCSQ(L,J),J=1,8)
 1815 FORMAT(8I8//)
 1810 CONTINUE
 1825 DO 1900 I=1,8
      DO 1900 J=1,8
      OCCSQ(I,J)=-OCCSQ(I,J)
 1900 CONTINUE
      NTURN=1
      MNPHT=0
      IF(MTURN.EQ.-1) MNPHT=1
      CALL MOVES(MPC,OPC,OCCSQ,MVALPC,MATTSQ,MPOSMV,MPHMV,
     . MNPHT,OENP,MCAS,MCA,NTURN,0,MNT)
      IF(MNT.EQ.0) GO TO 2125
      NTURN=1
      ONPHT=0
      IF(MTURN.EQ.-1) ONPHT=1
      CALL CHECK(MPC,OPC,OCCSQ,OVALPC,OATTSQ,OPOSMV,OPHMV,
     . CNPHT,CENP,OCAS,OCA,NTURN,ONT,MPOSMV,MPHMV,MNPHT,
     . MVALPC,MATTSQ,MNT,MENP,MCAS,MCA)
      IF(OCAS.EQ.2) GO TO 2125
      IKA=OPC(16)/10
      ML=0
      DO 2105 JKA=2,4
      IF(MATTSQ(IKA,JKA).NE.0) ML=1
 2105 CONTINUE
      MR=0
      DO 2110 JKA=6,7
      IF(MATTSQ(IKA,JKA).NE.0) MR=1
 2110 CONTINUE
      IF(ML.EQ.0) GO TO 2111
      OCAS=1
      IF(OCA.EQ.-10) OCAS=2
      IF(MR.NE.0) OCAS=2
      GO TO 2125
 2111 OCAS=0
      IF(OCA.EQ.-10) OCAS=-1
      IF(MR.NE.0) OCAS=-1
 2125 CONTINUE
      NTURN=1
      MPAR(1)=0
      IF(MTURN.EQ.-1) MPAR(1)=1
 2200 CALL PARGEN(MPOSMV,MNT,MATTSQ,MPAR,MVALPC,OCCSQ,
     . MCA,MPC,OPC,NTURN,MPHMV,MNPHT)
 3000 RETURN
      END
      SUBROUTINE EXEC(OMV,MENP,OENP,OCAS,OCA,OCCSQ,MPC,OPC,
     . JVALPC,NTURN)
      IMPLICIT INTEGER (O)
      DIMENSION MPC(16),OPC(16),OCCSQ(8,8),JVALPC(16)
      IF(OMV.EQ.0) GO TO 3000
      IF(OMV.LT.100) GO TO 2000
      OPN=OMV/10000
      OPR=OMV-10000*OPN
      NEWPOS=OPR/100
      OPRPOS=OPR-100*NEWPOS
DETERMINE IF MOVE WILL AFFECT CASTLE POSSIBILITIES
      IF(OCA.GE.50) GO TO 2364
      IF(OCA.EQ.10) GO TO 2362
      IF(OCA.EQ.-10) GO TO 2363
      IF(OPN.EQ.16) OCA=50
      IF(OPN.EQ.13) OCA=10
      IF(OPN.EQ.14) OCA=-10
      GO TO 2364
 2362 IF(OPN.EQ.14) OCA=50
```

```
      GO TO 2364
 2363 IF(OPN.EQ.13) OCA=50
 2364 IF(OPN.GT.8) GO TO 1000
SET EN PASSANT FLAG IF APPLICABLE
      IF(JVALPC(OPN).EQ.90) GO TO 1000
      NK34=NEWPOS-OPRPOS
      NK34=IABS(NK34)
      IF(NK34.NE.20) GO TO 800
      OENP=NEWPOS+100*OPN
  800 IF(NEWPOS.GT.78) GO TO 900
DETERMINE PAWN PROMOTION
      IF(NEWPOS.LT.21) GO TO 900
      GO TO 1000
  900 JVALPC(OPN)=90
IMPLEMENT MOVE OTHER THAN CASTLE
 1000 IOLD=OPRPOS/10
      JOLD=OPRPOS-10*IOLD
      INEW=NEWPOS/10
      JNEW=NEWPOS-10*INEW
      OCCSQ(IOLD,JOLD)=0
      IF(OCCSQ(INEW,JNEW).EQ.0) GO TO 1400
 1200 DO 1300 K=1,16
      IF(MPC(K).EQ.NEWPOS) MPC(K)=0
 1300 CONTINUE
      GO TO 1500
 1400 IF(OPN.GT.8) GO TO 1500
      IF(JVALPC(OPN).EQ.90) GO TO 1500
      IF(JOLD.EQ.JNEW) GO TO 1500
      OCCSQ(IOLD,JNEW)=0
      I25=MENP/100
 1450 MPC(I25)=0
 1500 OCCSQ(INEW,JNEW)=JVALPC(OPN)*NTURN
      OPC(OPN)=NEWPOS
      GO TO 3000
IMPLEMENT CASTLE MOVE
 2000 OCA=100
      OCAS=2
 2400 IF(OMV.EQ.99) GO TO 2500
      IR=OPC(13)/10
      JR=OPC(13)-10*IR
      KR=13
      K1=1
      GO TO 2600
 2500 IR=OPC(14)/10
      JR=OPC(14)-10*IR
      KR=14
      K1=-1
 2600 IK=OPC(16)/10
      JK=OPC(16)-10*IK
 2700 OCCSQ(IK,JK)=0
      OCCSQ(IR,JR)=0
      JR=JR+K1
      OCCSQ(IR,JR)=9900*NTURN
      JK=JR+K1
      OCCSQ(IK,JK)=50*NTURN
 2800 OPC(KR)=10*IK+JK
      OPC(16)=10*IR+JR
 3000 RETURN
      END
      SUBROUTINE CHECK(MPC,CPC,OCCSQ,OVALPC,OATTSQ,OPOSMV,
     . OPHMV,ONPHT,OENP,OCAS,OCA,NTURN,ONT,MPOSMV,MPHMV,
     . MNPHT,MVALPC,MATTSQ,MNT,MENP,MCAS,MCA)
      IMPLICIT INTEGER (O)
      DIMENSION MPC(16),OPC(16),OCCSQ(8,8),OVALPC(16),
     . OATTSQ(8,8),OPOSMV(75),OPHMV(75),MPOSMV(75),
     . MPHMV(75),JMPC(16),JOPC(16),JOCCSQ(8,8),JMVLPC(16),
     . MATTSQ(8,8),MVALPC(16)
      DO 50 I=1,8
      DO 50 J=1,8
      OCCSQ(I,J)=OCCSQ(I,J)*NTURN
   50 CONTINUE
      IF(MNT.EQ.0) GO TO 3100
```

```
      IFOG=0
      IF(ONPHT.NE.0) IFOG=1
      DO 1000 N=1,MNT
SAVE POSITION
      M1=MCAS
      M2=OCAS
      M3=MENP
      M4=OENP
      M5=MCA
      M6=OCA
      DO 100 K=1,16
      JMPC(K)=MPC(K)
      JOPC(K)=OPC(K)
      JMVLPC(K)=MVALPC(K)
  100 CONTINUE
      DO 200 I=1,8
      DO 200 J=1,8
      JOCCSQ(I,J)=OCCSQ(I,J)
  200 CONTINUE
      MP=MPOSMV(N)/100
      MMV=100*MPOSMV(N)+MPC(MP)
      IF(MPOSMV(N).LT.100) MMV=MPOSMV(N)
      NTURN=-1
      IF(IFOG.EQ.0) NTURN=1
EXECUTE MOVE
      CALL EXEC(MMV,OENP,MENP,MCAS,MCA,OCCSQ,OPC,MPC,
     . MVALPC,NTURN)
      KPN=0
      DO 2129 K=1,16
      IF(OPC(K).EQ.JOPC(K)) GO TO 2129
      KPN=K
      OSV=OVALPC(K)
      OVALPC(K)=0
 2129 CONTINUE
      NCHK=1
      NTURN=1
      ONPHT=0
      IF(IFOG.NE.0) ONPHT=1
      IF(NTURN.NE.1) GO TO 2348
      DO 2357 I=1,8
      DO 2357 J=1,8
      OCCSQ(I,J)=-OCCSQ(I,J)
 2357 CONTINUE
 2348 CONTINUE
      CALL MOVES(OPC,MPC,OCCSQ,OVALPC,OATTSQ,OPOSMV,
     . CPHMV,ONPHT,MENP,OCAS,OCA,NTURN,NCHK,ONT)
      IF(NTURN.NE.1) GO TO 2361
      DO 2368 I=1,8
      DO 2368 J=1,8
      OCCSQ(I,J)=-OCCSQ(I,J)
 2368 CONTINUE
 2361 CONTINUE
      MCAS=M1
      OCAS=M2
      MENP=M3
      OENP=M4
      MCA=M5
      OCA=M6
      DO 300 K=1,16
      OPC(K)=JOPC(K)
      MVALPC(K)=JMVLPC(K)
  300 CONTINUE
      IF(KPN.NE.0) OVALPC(KPN)=OSV
      DO 350 K=1,16
      IF(KPN.EQ.0) GO TO 310
      IF(MPC(K).EQ.OPC(KPN)) KPN=0
  310 MPC(K)=JMPC(K)
  350 CONTINUE
      DO 400 I=1,8
      DO 400 J=1,8
      OCCSQ(I,J)=JOCCSQ(I,J)
  400 CONTINUE
```

```
      IF(KPN.EQ.0) GO TO 500
      KPM=OPC(KPN)
      I=KPM/10
      J=KPM-10*I
      MATTSQ(I,J)=MATTSQ(I,J)-1001
DETERMINE IF MOVE RESULTS IN CHECK
  500 IF(NCHK.NE.1) GO TO 950
IF CHECK, CHANGE ATTSQ,POSMV
      IF(MPOSMV(N).LT.100) GO TO 990
      MPO=MPOSMV(N)-MP*100
      MI=MPO/10
      MJ=MPO-MI*10
      MATTSQ(MI,MJ)=MATTSQ(MI,MJ)-100*MVALPC(MP)-1
      IF(MVALPC(MP).EQ.32) GO TO 600
      IF(MVALPC(MP).EQ.50) GO TO 600
      IF(MVALPC(MP).EQ.90) GO TO 600
      GO TO 990
  600 DO 610 L=1,MNPHT
      M14=MPHMV(L)/100
      IF(M14.EQ.0) GO TO 610
      IF(MVALPC(M14).EQ.32) GO TO 603
      IF(MVALPC(M14).EQ.50) GO TO 603
      IF(MVALPC(M14).EQ.90) GO TO 603
      GO TO 610
  603 NVD=MPHMV(L)-100*M14
      IF(NVD.EQ.MPO) GO TO 604
      GO TO 610
  604 MPM=100*M14+MPC(MP)
      DO 607 K=1,MNPHT
      IF(MPHMV(K).EQ.MPM) GO TO 606
      GO TO 607
  606 MPHMV(L)=0
      GO TO 990
  607 CONTINUE
      GO TO 990
  610 CONTINUE
      GO TO 990
  950 IF(MVALPC(MP).NE.10) GO TO 1000
      I1=MPC(MP)/10
      J1=MPC(MP)-10*I1
      MPO=MPOSMV(N)-100*MP
      MI=MPO/10
      MJ=MPO-10*MI
      IF(J1.EQ.MJ) GO TO 1000
      DO 960 L=1,MNPHT
      M14=MPHMV(L)/100
      IF(M14.EQ.0) GO TO 960
      IF(MVALPC(M14).EQ.32) GO TO 952
      IF(MVALPC(M14).EQ.90) GO TO 952
      GO TO 960
  952 MPM=100*M14+MPC(MP)
      IF(MPHMV(L).NE.MPM) GO TO 960
      I4=-1
      IF(I1.GT.MI) I4=1
      J4=-1
      IF(J1.GT.MJ) J4=1
      I3=I1+I4
      J3=J1+J4
  954 IF(J3.LT.1) GO TO 960
      IF(J3.GT.8) GO TO 960
      IF(I3.LT.1) GO TO 960
      IF(I3.GT.8) GO TO 960
      N3=10*I3+J3
      IF(MPC(M14).EQ.N3) GO TO 965
      IF(OCCSQ(I3,J3).EQ.-90) GO TO 956
      IF(OCCSQ(I3,J3).EQ.-32) GO TO 956
      IF(OCCSQ(I3,J3).EQ.0) GO TO 956
      GO TO 960
  956 I3=I3+I4
      J3=J3+J4
      GO TO 954
  960 CONTINUE
```

```
      GO TO 1000
  965 MNPHT=MNPHT+1
      MPHMV(MNPHT)=100*M14+MPO
      MATTSQ(MI,MJ)=MATTSQ(MI,MJ)+100*MVALPC(M14)+1
      GO TO 1000
  990 MPOSMV(N)=0
 1000 CONTINUE
      DO 3000 N=1,MNPHT
SAVE POSITION
      IF(MPHMV(N).EQ.0) GO TO 3000
      M1=MCAS
      M2=OCAS
      M3=MENP
      M4=OENP
      DO 2100 K=1,16
      JMPC(K)=MPC(K)
      JOPC(K)=OPC(K)
      JMVLPC(K)=MVALPC(K)
 2100 CONTINUE
      DO 2200 I=1,8
      DO 2200 J=1,8
      JOCCSQ(I,J)=OCCSQ(I,J)
 2200 CONTINUE
EXECUTE PHANTOM MOVE
      MP=MPHMV(N)/100
      NEWPOS=MPHMV(N)-100*MP
      I9=MPC(MP)/10
      J9=MPC(MP)-10*I9
      IN=NEWPOS/10
      JN=NEWPOS-10*IN
      OCCSQ(I9,J9)=0
      OCCSQ(IN,JN)=MVALPC(MP)
      NCHK=1
      ONPHT=0
      IF(IFOG.NE.0) ONPHT=1
      IF(NTURN.NE.1) GO TO 2371
      DO 2369 I=1,8
      DO 2369 J=1,8
      OCCSQ(I,J)=-OCCSQ(I,J)
 2369 CONTINUE
 2371 CONTINUE
      CALL MOVES(OPC,MPC,OCCSQ,OVALPC,OATTSQ,OPOSMV,
     . OPHMV,ONPHT,MENP,OCAS,OCA,NTURN,NCHK,ONT)
      IF(NTURN.NE.1) GO TO 3381
      DO 3382 I=1,8
      DO 3382 J=1,8
      OCCSQ(I,J)=-OCCSQ(I,J)
 3382 CONTINUE
 3381 CONTINUE
      MCAS=M1
      OCAS=M2
      MENP=M3
      OENP=M4
      DO 2300 K=1,16
      MPC(K)=JMPC(K)
      OPC(K)=JOPC(K)
      MVALPC(K)=JMVLPC(K)
 2300 CONTINUE
      DO 2400 I=1,8
      DO 2400 J=1,8
      OCCSQ(I,J)=JOCCSQ(I,J)
 2400 CONTINUE
DETERMINE IF MOVE RESULTS IN CHECK
IF IN CHECK, MODIFY ATTSQ,PHMV
      IF(NCHK.NE.1) GO TO 3000
      OKSQ=OPC(16)
      IF(NEWPOS.EQ.OKSQ) GO TO 3000
      MATTSQ(IN,JN)=MATTSQ(IN,JN)-100*MVALPC(MP)-1
      MPHMV(N)=0
 3000 CONTINUE
      DO 3020 N=1,MNPHT
      MP=MPHMV(N)/100
```

```
      IF(MP.EQ.0) GO TO 3020
      IF(MVALPC(MP).NE.10) GO TO 3020
      I1=MPC(MP)/10
      J1=MPC(MP)-10*I1
      MPO=MPHMV(N)-100*MP
      MI=MPO/10
      MJ=MPO-10*MI
      IF(J1.EQ.MJ) GO TO 3020
      DO 3005 L=1,MNPHT
      M14=MPHMV(L)/100
      IF(M14.EQ.0) GO TO 3005
      IF(MVALPC(M14).EQ.32) GO TO 3002
      IF(MVALPC(M14).EQ.90) GO TO 3002
      GO TO 3005
 3002 MPM=100*M14+MPC(MP)
      IF(MPHMV(L).NE.MPM) GO TO 3005
      I4=-1
      IF(I1.GT.MI) I4=1
      J4=-1
      IF(J1.GT.MJ) J4=1
      I3=I1+I4
      J3=J1+J4
 3003 IF(I3.LT.1) GO TO 3005
      IF(I3.GT.8) GO TO 3005
      IF(J3.LT.1) GO TO 3005
      IF(J3.GT.8) GO TO 3005
      N3=10*I3+J3
      IF(MPC(M14).EQ.N3) GO TO 3010
      IF(OCCSQ(I3,J3).EQ.-90) GO TO 3004
      IF(OCCSQ(I3,J3).EQ.-32) GO TO 3004
      IF(OCCSQ(I3,J3).EQ.0) GO TO 3004
      GO TO 3005
 3004 I3=I3+I4
      J3=J3+J4
      GO TO 3003
 3005 CONTINUE
      GO TO 3020
 3010 MNPHT=MNPHT+1
      MPHMV(MNPHT)=100*M14+MPO
      MATTSQ(MI,MJ)=MATTSQ(MI,MJ)+100*MVALPC(M14)+1
 3020 CONTINUE
      M=0
      DO 3050 N=1,MNT
      IF(MPOSMV(N).NE.0) GO TO 3025
      GO TO 3050
 3025 M=M+1
      OPHMV(M)=MPOSMV(N)
 3050 CONTINUE
      MNT=M
      DO 3100 N=1,MNT
      MPOSMV(N)=OPHMV(N)
 3100 CONTINUE
      DO 3200 I=1,8
      DO 3200 J=1,8
      OCCSQ(I,J)=OCCSQ(I,J)*NTURN
 3200 CONTINUE
      RETURN
      END
      SUBROUTINE PARGEN(JPOSMV,JNT,JATTSQ,JPAR,JVALPC,
     . OCCSQ,JCA,JPC,KPC,NTURN,JPHMV,JNPHT)
      IMPLICIT INTEGER(O)
      DIMENSION JPOSMV(75),JATTSQ(8,8),JPAR(20),JVALPC(16),
     . OCCSQ(8,8),JPC(16),KPC(16),KATTSQ(8,8),KPHMV(75),
     . JV(8),KV(8),JPHMV(75),NWT(20),KVALPC(16),KPOSMV(75)
      NFR=0
      IF(NTURN.EQ.-1) NFR=1
      IF(JPAR(1).NE.0) NTURN=-NTURN
      NFOG=1
      IF(JPAR(1).NE.0) NFOG=-1
      DO 10 K=1,20
      JPAR(K)=0
   10 CONTINUE
```

```
      IF(JNT.EQ.0) GO TO 450
PAR(1)
      JPAR(1)=JNT
PAR(2)
      DO 75 K=1,15
      JPAR(2)=JPAR(2)+JVALPC(K)
   75 CONTINUE
PAR(3)
      DO 100 K=1,8
      IF(JPC(K).EQ.0) GO TO 100
      IF(JVALPC(K).EQ.90) GO TO 100
      IF(NTURN.EQ.-1) GO TO 85
      IMP=(JPC(K)/10)-2
      GO TO 90
   85 IMP=7-(JPC(K)/10)
   90 JPAR(3)=JPAR(3)+IMP**2
  100 CONTINUE
PAR(4)
      KCEN=44
      IF(NTURN.EQ.-1) KCEN=54
      KCENT=KCEN+1
      DO 110 K=1,8
      IF(JVALPC(K).EQ.90) GO TO 110
      IF(JPC(K).EQ.KCEN) GO TO 104
      IF(JPC(K).EQ.KCENT) GO TO 106
      GO TO 110
  104 JPAR(4)=JPAR(4)+10
      KTR=KCENT-10*NTURN
      DO 105 L=1,8
      IF(JVALPC(L).EQ.90) GO TO 105
      IF(JPC(L).EQ.KTR) JPAR(4)=JPAR(4)+10
  105 CONTINUE
      GO TO 110
  106 JPAR(4)=JPAR(4)+10
      KTR=KCEN-10*NTURN
      DO 107 L=1,8
      IF(JVALPC(L).EQ.90) GO TO 107
      IF(JPC(L).EQ.KTR) JPAR(4)=JPAR(4)+10
  107 CONTINUE
  110 CONTINUE
      DO 120 K=1,8
      IF(KPC(K).EQ.KCEN) GO TO 111
      IF(KPC(K).EQ.KCENT) GO TO 113
      GO TO 120
  111 KTR=KCEN-11*NTURN
      K35=0
  112 DO 115 L=1,8
      IF(JPC(L).EQ.KTR) JPAR(4)=JPAR(4)+10
  115 CONTINUE
      IF(K35.NE.0) GO TO 120
      KTR=KCEN-9*NTURN
      K35=1
      GO TO 112
  113 KTR=KCENT-11*NTURN
      K35=0
  114 DO 116 L=1,8
      IF(JPC(L).EQ.KTR) JPAR(4)=JPAR(4)+10
  116 CONTINUE
      IF(K35.NE.0) GO TO 120
      KTR=KCENT-9*NTURN
      K35=1
      GO TO 114
  120 CONTINUE
PAR(5)
      IK=KPC(16)/10
      JK=KPC(16)-10*IK
      IJK=JPC(16)/10
      JJK=JPC(16)-10*IJK
      IF(JNT.EQ.1) JPAR(5)=JPAR(5)-1
      M14=0
      DO 130 K=1,JNT
      NVD=JPOSMV(K)/100
```

```
      IF(NVD.NE.16) M14=1
      IF(NVD.NE.16) GO TO 135
  130 CONTINUE
  135 IF(M14.EQ.0) JPAR(5)=JPAR(5)-1
PAR(6)
      I=IJK+NTURN
      IF(I.GT.8) GO TO 140
      IF(I.LT.1) GO TO 140
      DO 140 K=1,3
      J=JJK+K-2
      IF(J.LT.1) GO TO 140
      IF(J.GT.8) GO TO 140
      IF(OCCSQ(I,J).GT.0) JPAR(6)=JPAR(6)+10
  140 CONTINUE
PAR(7)
      IF(JCA.EQ.100) JPAR(7)=200
      IF(JCA.EQ.100) GO TO 155
      DO 150 N=1,JNT
      IF(JPOSMV(N).EQ.98) JPAR(7)=100
      IF(JPOSMV(N).EQ.99) JPAR(7)=100
  150 CONTINUE
PAR(8)
  155 DO 170 K=1,4
      L=K+8
      IF(JPC(L).EQ.0) GO TO 170
      IF(NTURN.EQ.-1) GO TO 160
      IF(JPC(L).GT.18) JPAR(8)=JPAR(8)+10
      GO TO 170
  160 IF(JPC(L).LT.81) JPAR(8)=JPAR(8)+10
  170 CONTINUE
      IF(JPC(15).EQ.0) GO TO 195
      IF(NTURN.EQ.-1) GO TO 180
      IF(JPC(15).GT.18) JPAR(8)=JPAR(8)+10
      GO TO 190
  180 IF(JPC(15).LT.81) JPAR(8)=JPAR(8)+10
  190 CONTINUE
PAR(9)
  195 DO 210 J=1,8
      IF(JATTSQ(IK,J).EQ.0) GO TO 210
      NVD=10*IK+J
      JAT=JATTSQ(IK,J)/100
      JAK=JATTSQ(IK,J)-100*JAT
      IF(JAK.NE.1) GO TO 201
      IF(JAT.EQ.50) JPAR(9)=JPAR(9)+10
      IF(JAT.EQ.90) JPAR(9)=JPAR(9)+10
      GO TO 210
  201 JPAR(9)=JPAR(9)+10
      IF(OCCSQ(IK,J).GT.0) GO TO 205
      DO 204 N=1,JNT
      M14=JPOSMV(N)/100
      IF(M14.EQ.0) GO TO 204
      IF(JVALPC(M14).EQ.50) GO TO 202
      IF(JVALPC(M14).EQ.90) GO TO 202
      GO TO 204
  202 LP2=JPOSMV(N)-100*M14
      IF(LP2.EQ.NVD) JPAR(9)=JPAR(9)+10
  204 CONTINUE
  205 DO 208 N=1,JNPHT
      M14=JPHMV(N)/100
      IF(JVALPC(M14).EQ.50) GO TO 206
      IF(JVALPC(M14).EQ.90) GO TO 206
      GO TO 208
  206 LP2=JPHMV(N)-100*M14
      IF(LP2.EQ.NVD) JPAR(9)=JPAR(9)+10
  208 CONTINUE
  210 CONTINUE
      I=IK-NTURN
      IF(I.GT.8) GO TO 219
      IF(I.LT.1) GO TO 219
      DO 219 K=1,3
      NAD=10
      IF(K.EQ.2) NAD=5
```

```
      J=JK+K-2
      IF(J.LT.1) GO TO 219
      IF(J.GT.8) GO TO 219
      NVD=10*I+J
      IF(JATTSQ(I,J).EQ.0) GO TO 219
      JAT=JATTSQ(I,J)/100
      JAK=JATTSQ(I,J)-100*JAT
      IF(JAK.NE.1) GO TO 211
      IF(JAT.EQ.32) JPAR(9)=JPAR(9)+NAD
      IF(JAT.EQ.90) JPAR(9)=JPAR(9)+NAD
      IF(JAT.EQ.50) JPAR(9)=JPAR(9)+NAD
      GO TO 219
  211 JPAR(9)=JPAR(9)+NAD
      IF(OCCSQ(I,J).GT.0) GO TO 215
      DO 214 N=1,JNT
      M14=JPOSMV(N)/100
      IF(M14.EQ.0) GO TO 214
      IF(JVALPC(M14).EQ.32) GO TO 212
      IF(JVALPC(M14).EQ.90) GO TO 212
      IF(JVALPC(M14).EQ.50) GO TO 212
      GO TO 214
  212 LP2=JPOSMV(N)-100*M14
      IF(LP2.EQ.NVD) JPAR(9)=JPAR(9)+NAD
  214 CONTINUE
  215 DO 218 N=1,JNPHT
      M14=JPHMV(N)/100
      IF(JVALPC(M14).EQ.32) GO TO 216
      IF(JVALPC(M14).EQ.90) GO TO 216
      IF(JVALPC(M14).EQ.50) GO TO 216
      GO TO 218
  216 LP2=JPHMV(N)-100*M14
      IF(LP2.EQ.NVD) JPAR(9)=JPAR(9)+NAD
  218 CONTINUE
  219 CCNTINUE
PAR(10)
      DO 220 K=1,15
      IF(KPC(K).EQ.0) GO TO 220
      IK=KPC(K)/10
      JK=KPC(K)-10*IK
      IF(JATTSQ(IK,JK).GT.0) JPAR(10)=JPAR(10)+10
  220 CONTINUE
PAR(11)
      DO 224 K=1,8
      IF(JPC(K).EQ.0) GO TO 224
      IF(JPC(K).EQ.90) GO TO 224
      I=JPC(K)/10
      J=JPC(K)-10*I
      DO 222 L=1,5
      I=I+NTURN
      IF(I.GT.7) GO TO 224
      IF(I.LT.2) GO TO 224
      IF(OCCSQ(I,J).EQ.-10) JPAR(11)=JPAR(11)-10
      IF(OCCSQ(I,J).EQ.-10) GO TO 224
  222 CONTINUE
  224 CONTINUE
      IF(NFR.NE.1) GO TO 230
      DO 225 I=1,8
      DO 225 J=1,8
      KATTSQ(I,J)=JATTSQ(I,J)
  225 CONTINUE
      KNPHT=JNPHT
      DO 228 N=1,JNPHT
      KPHMV(N)=JPHMV(N)
  228 CONTINUE
      KNT=JNT
      DO 227 N=1,JNT
      KPOSMV(N)=JPOSMV(N)
  227 CONTINUE
      DO 229 K=1,16
      KVALPC(K)=JVALPC(K)
  229 CONTINUE
      GO TO 400
```

```
PAR(12)
  230 JWORST=0
      DO 300 K=1,15
      IF(KPC(K).EQ.0) GO TO 300
      IM=KPC(K)/10
      JM=KPC(K)-10*IM
      KAT=KATTSQ(IM,JM)-(KATTSQ(IM,JM)/10)*10
      IF(JATTSQ(IM,JM).NE.0) GO TO 235
      JPAR(12)=JPAR(12)+10*KAT
      GO TO 300
  235 JAT=JATTSQ(IM,JM)-(JATTSQ(IM,JM)/10)*10
      LOST=IABS(OCCSQ(IM,JM))
      IF(KAT.NE.0) GO TO 240
      IF(LOST.GT.JWORST) JWORST=LOST
      GO TO 300
  240 IF(JAT.NE.1) GO TO 245
      IF(JATTSQ(IM,JM).GT.900000) GO TO 244
      LGAIN=JATTSQ(IM,JM)/100
      IF(LOST.GT.LGAIN) GO TO 242
      JPAR(12)=JPAR(12)+LGAIN-LOST
      GO TO 300
  242 LOST=LOST-LGAIN
      IF(LOST.GT.JWORST) JWORST=LOST
      GO TO 300
  244 JPAR(12)=JPAR(12)+10
      GO TO 300
  245 IF(KAT.GE.JAT) GO TO 270
      DO 246 N=1,8
      JV(N)=0
  246 CONTINUE
      L=0
      DO 250 N=1,JNT
      KMV=JPOSMV(N)-(JPOSMV(N)/100)*100
      IF(KMV.NE.KPC(K)) GO TO 250
      KC=(JPOSMV(N)-KMV)/100
      I5=JPC(KC)/10
      J5=JPC(KC)-10*I5
      L=L+1
      JV(L)=IABS(OCCSQ(I5,J5))
  250 CONTINUE
      L=0
      KT5=JAT-1
      DO 255 M=1,KT5
      L=L+1
      KL=15000
      DO 253 N=L,JAT
      IF(JV(N).LE.KL) KL=JV(N)
      IF(JV(N).LE.KL) KN=N
  253 CONTINUE
      KSA=JV(L)
      JV(L)=KL
      JV(KN)=KSA
  255 CONTINUE
      IF(KATTSQ(IM,JM).GT.900000) GO TO 265
  259 LOST=IABS(OCCSQ(IM,JM))+KATTSQ(IM,JM)/100
      LGAIN=0
      DO 260 L=1,KAT
      LGAIN=LGAIN+JV(L)
  260 CONTINUE
      IF(LOST.GT.LGAIN) GO TO 262
      JPAR(12)=JPAR(12)+LGAIN-LOST.
      GO TO 300
  262 LOST=LOST-LGAIN
      IF(LOST.GT.JWORST) JWORST=LOST
      GO TO 300
  265 KAT=KAT-1
      GO TO 259
  270 DO 271 N=1,8
      KV(N)=0
  271 CONTINUE
      L=0
      DO 275 N=1,KNPHT
```

```
      KMV=KPHMV(N)-(KPHMV(N)/100)*100
      IF(KMV.NE.KPC(K)) GO TO 275
      KC=(KPHMV(N)-KMV)/100
      I5=KPC(KC)/10
      J5=KPC(KC)-10*I5
      L=L+1
      KV(L)=IABS(OCCSQ(I5,J5))
  275 CONTINUE
      L=0
      KT5=KAT-1
      DO 280 M=1,KT5
      L=L+1
      KL=15000
      DO 278 N=L,KAT
      IF(KV(N).LE.KL) KL=KV(N)
      IF(KV(N).LE.KL) KN=N
  278 CONTINUE
      KSA=KV(L)
      KV(L)=KL
      KV(KN)=KSA
  280 CONTINUE
      IF(JATTSQ(IM,JM).GT.900000) GO TO 290
  281 LGAIN=JATTSQ(IM,JM)/100
      LOST=IABS(OCCSQ(IM,JM))
      JAT=JAT-1
      DO 285 L=1,JAT
      LOST=LOST+KV(L)
  285 CONTINUE
      IF(LOST.GT.LGAIN) GO TO 288
      JPAR(12)=JPAR(12)+LGAIN-LOST
      GO TO 300
  288 LOST=LOST-LGAIN
      IF(LOST.GT.JWORST) JWORST=LOST
      GO TO 300
  290 JAT=JAT-1
      GO TO 281
  300 CONTINUE
      IF(JWORST.EQ.0) GO TO 400
      JPAR(12)=-100*JWORST
  400 JPAR(12)=-JPAR(12)/10
      NTWT=NFR*NFOG
      IF(NTWT.NE.1) GO TO 408
      READ(5,405) (NWT(K),K=1,17)
  405 FORMAT(17I4)
PAR(13)
  408 IF(NFR.EQ.1) GO TO 442
      DO 430 K=8,16
      I=KPC(K)/10
      I1=I-NTURN
      IF(I1.GT.6) GO TO 430
      IF(I1.LT.3) GO TO 430
      J=KPC(K)-10*I
      J1=J-1
      IF(J1.LT.1) GO TO 416
  414 NVD=10*I1+J1
      DO 410 L=1,JNT
      M14=JPOSMV(L)/100
      IF(M14.EQ.0) GO TO 410
      IF(M14.GT.8) GO TO 410
      IF(JVALPC(M14).NE.10) GO TO 410
      N2=JPOSMV(L)-100*M14
      IF(N2.EQ.NVD) GO TO 409
      GO TO 410
  409 IF(JATTSQ(I1,J1).NE.0) GO TO 412
      I2=I1
  413 I2=I2-NTURN
      IF(I2.GT.8) GO TO 417
      IF(I2.LT.1) GO TO 417
      IF(OCCSQ(I2,J1).EQ.0) GO TO 413
      IF(OCCSQ(I2,J1).EQ.10) GO TO 413
      IF(OCCSQ(I2,J1).EQ.90) GO TO 412
      IF(OCCSQ(I2,J1).EQ.50) GO TO 412
```

```
  417 IF(KATTSQ(I1,J1).NE.0) GO TO 416
      JPAR(13)=JPAR(13)+10
      GO TO 416
  412 N43=0
      DO 415 N=1,KNPHT
      LP2=KPHMV(N)/100
      IF(KVALPC(LP2).NE.10) GO TO 415
      N31=KPHMV(N)-100*LP2
      IF(N31.NE.NVD) GO TO 415
      N43=1
  415 CONTINUE
      IF(N43.EQ.0) JPAR(13)=JPAR(13)+10
  410 CONTINUE
  416 IF(J.LT.J1) GO TO 430
      J1=J+1
      IF(J.GT.8) GO TO 430
      GO TO 414
  430 CONTINUE
PAR(14)
      DO 475 J=1,8
      I=KPC(16)/10
      I1=I
  452 I=I-NTURN
      IF(I.GT.8) GO TO 475
      IF(I.LT.1) GO TO 475
      M14=10*I+J
      DO 455 N=1,JNT
      IF(JPOSMV(N).LT.100) GO TO 455
      NVD=JPOSMV(N)/100
      LP2=JPOSMV(N)-100*NVD
      IF(LP2.NE.M14) GO TO 455
      IF(JVALPC(NVD).EQ.90) GO TO 453
      IF(JVALPC(NVD).EQ.50) GO TO 453
      GO TO 455
  453 IF(KATTSQ(I,J).NE.0) GO TO 455
      J3=JPC(NVD)-(JPC(NVD)/10)*10
      IF(J3.EQ.J) GO TO 455
      JPAR(14)=JPAR(14)+10
      JAT=JATTSQ(I1,J)-(JATTSQ(I1,J)/100)*100
      JAT=JAT+1
      KAT=KATTSQ(I1,J)-(KATTSQ(I1,J)/100)*100
      IF(JAT.GT.KAT) JPAR(14)=JPAR(14)+50
  455 CONTINUE
      IF(OCCSQ(I,J).EQ.0) GO TO 452
      I5=IABS(I-I1)
      IF(I5.NE.1) GO TO 475
      J5=KPC(16)-10*I1
      DO 460 M=1,3
      J1=J5+M-2
      IF(J1.LT.1) J1=J1+3
      IF(J1.GT.8) J1=J1-3
      IF(J.EQ.J1) GO TO 452
  460 CONTINUE
  475 CONTINUE
PAR(15)
      DO 525 K=1,3
      I=KPC(16)/10
      J=KPC(16)-10*I
      I1=I-NTURN
      J1=J+K-2
      IF(J1.LT.1) J1=J1+3
      IF(J1.GT.8) J1=J1-3
      J5=-1
      I3=I1-NTURN
      J3=J1
  524 IF(I3.GT.8) GO TO 526
      IF(I3.LT.1) GO TO 526
      J3=J3+J5
      IF(J3.LT.1) GO TO 526
      IF(J3.GT.8) GO TO 526
      LP2=10*I3+J3
      DO 535 N=1,JNT
```

```
      M14=JPOSMV(N)/100
      NVD=JPOSMV(N)-100*M14
      IF(NVD.NE.LP2) GO TO 535
      IF(KATTSQ(I3,J3).NE.0) GO TO 535
      IF(JVALPC(M14).EQ.90) GO TO 529
      IF(JVALPC(M14).EQ.32) GO TO 529
      GO TO 535
  529 M18=0
      DO 531 M=1,JNT
      M15=JPOSMV(M)/100
      M16=JPOSMV(M)-100*M15
      IF(M15.NE.M14) GO TO 531
      M17=10*I1+J1
      IF(M16.EQ.M17) M18=1
  531 CONTINUE
      IF(M18.NE.0) GO TO 535
  534 JPAR(15)=JPAR(15)+10
      JAT=JATTSQ(I1,J1)-(JATTSQ(I1,J1)/100)*100
      JAT=JAT+1
      KAT=KATTSQ(I1,J1)-(KATTSQ(I1,J1)/100)*100
      IF(JAT.GT.KAT) JPAR(15)=JPAR(15)+50
  535 CONTINUE
      I3=I3-NTURN
      GO TO 524
  526 IF(J5.EQ.1) GO TO 525
      J5=1
      I3=I1-NTURN
      J3=J1
      GO TO 524
  525 CONTINUE
PAR(16)
      DO 575 J=1,8
      I=JPC(16)/10
      I1=I
  552 I=I+NTURN
      IF(I.GT.8) GO TO 575
      IF(I.LT.1) GO TO 575
      M14=10*I+J
      DO 555 N=1,KNT
      IF(KPOSMV(N).LT.100) GO TO 555
      NVD=KPOSMV(N)/100
      LP2=KPOSMV(N)-100*NVD
      IF(LP2.NE.M14) GO TO 555
      IF(KVALPC(NVD).EQ.90) GO TO 553
      IF(KVALPC(NVD).EQ.50) GO TO 553
      GO TO 555
  553 IF(JATTSQ(I,J).NE.0) GO TO 555
      J3=KPC(NVD)-(KPC(NVD)/10)*10
      IF(J3.EQ.J) GO TO 555
      JPAR(16)=JPAR(16)-10
      JAT=JATTSQ(I1,J)-(JATTSQ(I1,J)/100)*100
      KAT=KATTSQ(I1,J)-(KATTSQ(I1,J)/100)*100
      KAT=KAT+1
      IF(KAT.GT.JAT) JPAR(16)=JPAR(16)-50
  555 CONTINUE
      IF(OCCSQ(I,J).EQ.0) GO TO 552
      I5=IABS(I-I1)
      IF(I5.NE.1) GO TO 575
      J5=JPC(16)-10*I1
      DO 560 M=1,3
      J1=J5+M-2
      IF(J1.LT.1) J1=J1+3
      IF(J1.GT.8) J1=J1-3
      IF(J.EQ.J1) GO TO 552
  560 CONTINUE
  575 CONTINUE
PAR(17)
      DO 625 K=1,3
      I=JPC(16)/10
      J=JPC(16)-10*I
      I1=I+NTURN
      J1=J+K-2
```

```
      IF(J1.LT.1) J1=J1+3
      IF(J1.GT.8) J1=J1-3
      J5=-1
      I3=I1+NTURN
      J3=J1
  624 IF(I3.GT.8) GO TO 626
      IF(I3.LT.1) GO TO 626
      J3=J3+J5
      IF(J3.LT.1) GO TO 626
      IF(J3.GT.8) GO TO 626
      LP2=10*I3+J3
      DO 635 N=1,KNT
      M14=KPOSMV(N)/100
      NVD=KPOSMV(N)-100*M14
      IF(NVD.NE.LP2) GO TO 635
      IF(JATTSQ(I3,J3).NE.0) GO TO 635
      IF(KVALPC(M14).EQ.90) GO TO 629
      IF(KVALPC(M14).EQ.32) GO TO 629
      GO TO 635
  629 M18=0
      DO 631 M=1,KNT
      M15=KPOSMV(M)/100
      M16=KPOSMV(M)-100*M15
      IF(M15.NE.M14) GO TO 631
      M17=10*I1+J1
      IF(M16.EQ.M17) M18=1
  631 CONTINUE
      IF(M18.NE.0) GO TO 635
  634 JPAR(17)=JPAR(17)-10
      JAT=JATTSQ(I1,J1)-(JATTSQ(I1,J1)/100)*100
      KAT=KATTSQ(I1,J1)-(KATTSQ(I1,J1)/100)*100
      KAT=KAT+1
      IF(KAT.GT.JAT) JPAR(17)=JPAR(17)-50
  635 CONTINUE
      I3=I3+NTURN
      GO TO 624
  626 IF(J5.EQ.1) GO TO 625
      J5=1
      I3=I1+NTURN
      J3=J1
      GO TO 624
  625 CONTINUE
PAR(20)
  442 DO 444 K=1,16
      JPAR(20)=JPAR(20)+JPAR(K)*NWT(K)
  444 CONTINUE
  450 RETURN
      END
      SUBROUTINE MOVES(MPC,OPC,OCCSQ,JVALPC,MATTSQ,MPOSMV,
     . MPHMV,MNRHT,OENP,MCAS,MCA,NTURN,NCHK,MNT)
      IMPLICIT INTEGER (O)
      DIMENSION MPC(16),OPC(16),OCCSQ(8,8),JVALPC(16),
     . MATTSQ(8,8),MPOSMV(75),MPHMV(75)
      DO 50 I=1,8
      DO 50 J=1,8
      MATTSQ(I,J)=0
      OCCSQ(I,J)=OCCSQ(I,J)*NTURN
   50 CONTINUE
      N=0
      NN=0
      IF(NTURN.EQ.-1) GO TO 65
      KINGSQ=OPC(16)
      GO TO 70
   65 KINGSQ=MPC(16)
   70 IF(NCHK.EQ.0) KINGSQ=0
      NCHK=0
      DO 2000 K=1,16
      IF(NTURN.EQ.-1) GO TO 75
      IF(MPC(K).EQ.0) GO TO 2000
      NPOS=MPC(K)
      GO TO 100
   75 IF(OPC(K).EQ.0) GO TO 2000
```

```
      NPOS=OPC(K)
  100 IP=NPOS/10
      JP=NPOS-10*IP
      IF(K.GT.8) GO TO 1100
      IF(JVALPC(K).EQ.90) GO TO 1400
PAWN MOVES
      IF(MNPHT.NE.0) NTURN=-NTURN
      I=IP+NTURN
      IF(OCCSQ(I,JP).NE.0) GO TO 1010
      N=N+1
      MPOSMV(N)=K*100+10*I+JP
      NCP=NTURN*(2*IP-9)+5
      IF(NCP.NE.0) GO TO 1010
      I=I+NTURN
      IF(OCCSQ(I,JP).NE.0) GO TO 1010
      N=N+1
      MPOSMV(N)=K*100+10*I+JP
 1010 I=IP+NTURN
      IF(MNPHT.NE.0) NTURN=-NTURN
      J=JP-1
      IF(J.LE.0) GO TO 1030
 1020 MATTSQ(I,J)=MATTSQ(I,J)+100*JVALPC(K)+1
      NN=NN+1
      MPHMV(NN)=K*100+10*I+J
      MSQ=10*I+J
      IF(MSQ.EQ.KINGSQ) GO TO 2400
      IF(OCCSQ(I,J).GE.0) GO TO 1025
      IF(OCCSQ(I,J).EQ.-9900) GO TO 1025
      N=N+1
      MPOSMV(N)=K*100+10*I+J
      IF(MPOSMV(N).EQ.MPHMV(NN)) NN=NN-1
 1025 IF(J.GT.JP) GO TO 2000
 1030 J=JP+1
      IF(J.GE.9) GO TO 2000
      GO TO 1020
KNIGHT MOVES
 1100 IF(K.GT.10) GO TO 1200
      I=IP+1
      J=JP+2
      ASSIGN 1110 TO MAR
      GO TO 1185
 1110 J=JP-2
      ASSIGN 1120 TO MAR
      GO TO 1185
 1120 I=IP+2
      J=JP+1
      ASSIGN 1130 TO MAR
      GO TO 1185
 1130 J=JP-1
      ASSIGN 1140 TO MAR
      GO TO 1185
 1140 I=IP-1
      J=JP-2
      ASSIGN 1150 TO MAR
      GO TO 1185
 1150 J=JP+2
      ASSIGN 1160 TO MAR
      GO TO 1185
 1160 I=IP-2
      J=JP-1
      ASSIGN 1170 TO MAR
      GO TO 1185
 1170 J=JP+1
      ASSIGN 2000 TO MAR
 1185 NKC=I*J
      IF(NKC.LE.0) GO TO 1189
      IF(I.GT.8) GO TO 1189
      IF(J.GT.8) GO TO 1189
      MATTSQ(I,J)=MATTSQ(I,J)+100*JVALPC(K)+1
      NN=NN+1
      MPHMV(NN)=K*100+10*I+J
      MSQ=10*I+J
```

```
      IF(MSQ.EQ.KINGSQ) GO TO 2400
      IF(OCCSQ(I,J).GT.0) GO TO 1189
      IF(OCCSQ(I,J).EQ.-9900) GO TO 1189
      N=N+1
      MPOSMV(N)=100*K+10*I+J
      IF(MPOSMV(N).EQ.MPHMV(NN)) NN=NN-1
 1189 GO TO MAR,(1110,1120,1130,1140,1150,1160,1170,2000)
BISHOP MOVES
 1200 IF(K.GT.12) GO TO 1300
      NQ=0
 1201 NDB=0
      ND=0
      I=IP
      J=JP
 1202 I=I+1
      IF(I.GT.8) GO TO 1210
      J=J+1
      IF(J.GT.8) GO TO 1210
      IF(OCCSQ(I,J).EQ.0) GO TO 1205
      IF(OCCSQ(I,J).LT.0) NDB=0
      IF(OCCSQ(I,J).EQ.90) NDB=1
      IF(OCCSQ(I,J).EQ.80) NDB=1
      IF(OCCSQ(I,J).EQ.32) NDB=1
      IF(NDB.EQ.1) ND=1
      IF(NDB.EQ.1) GO TO 1205
      ASSIGN 1210 TO MAR
      GO TO 1280
 1205 ASSIGN 1202 TO MAR
      GO TO 1280
 1210 NDB=0
      ND=0
      I=IP
      J=JP
 1215 I=I-1
      IF(I.LE.0) GO TO 1220
      J=J-1
      IF(J.LE.0) GO TO 1220
      IF(OCCSQ(I,J).EQ.0) GO TO 1218
      IF(OCCSQ(I,J).LT.0) NDB=0
      IF(OCCSQ(I,J).EQ.90) NDB=1
      IF(OCCSQ(I,J).EQ.80) NDB=1
      IF(OCCSQ(I,J).EQ.32) NDB=1
      IF(NDB.EQ.1) ND=1
      IF(NDB.EQ.1) GO TO 1218
      ASSIGN 1220 TO MAR
      GO TO 1280
 1218 ASSIGN 1215 TO MAR
      GO TO 1280
 1220 NDB=0
      ND=0
      I=IP
      J=JP
 1225 I=I+1
      IF(I.GT.8) GO TO 1230
      J=J-1
      IF(J.LE.0) GO TO 1230
      IF(OCCSQ(I,J).EQ.0) GO TO 1228
      IF(OCCSQ(I,J).LT.0) NDB=0
      IF(OCCSQ(I,J).EQ.90) NDB=1
      IF(OCCSQ(I,J).EQ.80) NDB=1
      IF(OCCSQ(I,J).EQ.32) NDB=1
      IF(NDB.EQ.1) ND=1
      IF(NDB.EQ.1) GO TO 1228
      ASSIGN 1230 TO MAR
      GO TO 1280
 1228 ASSIGN 1225 TO MAR
      GO TO 1280
 1230 NDB=0
      ND=0
      I=IP
      J=JP
 1235 I=I-1
```

```
      IF(I.LE.0) GO TO 1290
      J=J+1
      IF(J.GT.8) GO TO 1290
      IF(OCCSQ(I,J).EQ.0) GO TO 1238
      IF(OCCSQ(I,J).LT.0) NDB=0
      IF(OCCSQ(I,J).EQ.90) NDB=1
      IF(OCCSQ(I,J).EQ.80) NDB=1
      IF(OCCSQ(I,J).EQ.32) NDB=1
      IF(NDB.EQ.1) ND=1
      IF(NDB.EQ.1) GO TO 1238
      ASSIGN 1290 TO MAR
      GO TO 1280
 1238 ASSIGN 1235 TO MAR
 1280 IF(OCCSQ(I,J).GT.0) GO TO 1285
      IF(OCCSQ(I,J).EQ.-9900) GO TO 1285
      IF(ND.EQ.1) GO TO 1285
      N=N+1
      MPCSMV(N)=100*K+10*I+J
 1285 MATTSQ(I,J)=MATTSQ(I,J)+100*JVALPC(K)+1
      NN=NN+1
      MPHMV(NN)=K*100+10*I+J
      IF(MPOSMV(N).EQ.MPHMV(NN)) NN=NN-1
      MSQ=10*I+J
      IF(MSQ.EQ.KINGSQ) GO TO 2400
      NDB=0
      GO TO MAR,(1210,1202,1220,1215,1230,1225,1290,1235)
 1290 IF(NQ.EQ.0) GO TO 2000
      GO TO 1301
ROCK MOVES
 1300 IF(K.GT.14) GO TO 1400
 1301 NDR=0
      ND=0
      I=IP
      J=JP
 1305 I=I+1
      IF(I.GT.8) GO TO 1310
      IF(OCCSQ(I,J).EQ.0) GO TO 1308
      IF(OCCSQ(I,J).LT.0) NDR=0
      IF(OCCSQ(I,J).EQ.50) NDR=1
      IF(OCCSQ(I,J).EQ.90) NDR=1
      IF(OCCSQ(I,J).EQ.80) NDR=1
      IF(NDR.EQ.1) ND=1
      IF(NDR.EQ.1) GO TO 1308
      ASSIGN 1310 TO MAR
      GO TO 1380
 1308 ASSIGN 1305 TO MAR
      GO TO 1380
 1310 NDR=0
      ND=0
      I=IP
 1315 I=I-1
      IF(I.LE.0) GO TO 1320
      IF(OCCSQ(I,J).EQ.0) GO TO 1318
      IF(OCCSQ(I,J).LT.0) NDR=0
      IF(OCCSQ(I,J).EQ.50) NDR=1
      IF(OCCSQ(I,J).EQ.90) NDR=1
      IF(OCCSQ(I,J).EQ.80) NDR=1
      IF(NDR.EQ.1) ND=1
      IF(NDR.EQ.1) GO TO 1318
      ASSIGN 1320 TO MAR
      GO TO 1380
 1318 ASSIGN 1315 TO MAR
      GO TO 1380
 1320 NDR=0
      ND=0
      I=IP
      J=JP
 1325 J=J+1
      IF(J.GT.8) GO TO 1330
      IF(OCCSQ(I,J).EQ.0) GO TO 1328
      IF(OCCSQ(I,J).LT.0) NDR=0
      IF(OCCSQ(I,J).EQ.50) NDR=1
```

```
      IF(OCCSQ(I,J).EQ.90) NDR=1
      IF(OCCSQ(I,J).EQ.80) NDR=1
      IF(NDR.EQ.1) ND=1
      IF(NDR.EQ.1) GO TO 1328
      ASSIGN 1330 TO MAR
      GO TO 1380
 1328 ASSIGN 1325 TO MAR
      GO TO 1380
 1330 NDR=0
      ND=0
      I=IP
      J=JP
 1335 J=J-1
      IF(J.LE.0) GO TO 1390
      IF(OCCSQ(I,J).EQ.0) GO TO 1338
      IF(OCCSQ(I,J).LT.0) NDR=0
      IF(OCCSQ(I,J).EQ.50) NDR=1
      IF(OCCSQ(I,J).EQ.90) NDR=1
      IF(OCCSQ(I,J).EQ.80) NDR=1
      IF(NDR.EQ.1) ND=1
      IF(NDR.EQ.1) GO TO 1338
      ASSIGN 1390 TO MAR
      GO TO 1380
 1338 ASSIGN 1335 TO MAR
 1380 IF(OCCSQ(I,J).GT.0) GO TO 1385
      IF(OCCSQ(I,J).EQ.-9900) GO TO 1385
      IF(ND.EQ.1) GO TO 1385
      N=N+1
      MPCSMV(N)=100*K+10*I+J
 1385 MATTSQ(I,J)=MATTSQ(I,J)+100*JVALPC(K)+1
      NN=NN+1
      MPHMV(NN)=K*100+10*I+J
      IF(MPOSMV(N).EQ.MPHMV(NN)) NN=NN-1
      IF(OCCSQ(I,J).NE.9900) GO TO 1388
      IF(ND.EQ.1) GO TO 1388
CASTLE MOVES
      IF(K.NE.13) GO TO 1386
      IF(MCAS.GT.0) GO TO 1388
      IF(MCA.GE.50) GO TO 1388
      N=N+1
      MPCSMV(N)=98
 1386 IF(K.NE.14) GO TO 1388
      IF(MCAS.LT.0) GO TO 1388
      IF(MCAS.GT.1) GO TO 1388
      IF(MCA.GE.50) GO TO 1388
      N=N+1
      MPOSMV(N)=99
 1388 MSQ=10*I+J
      IF(MSQ.EQ.KINGSQ) GO TO 2400
      NDR=0
      GO TO MAR,(1310,1305,1320,1315,1330,1325,1390,1335)
 1390 IF(NQ.EQ.0) GO TO 2000
      GO TO 1401
QUEEN MOVES
 1400 IF(K.GT.15) GO TO 1500
      NQ=1
      GO TO 1201
 1401 NQ=0
      GO TO 2000
KING MOVES
 1500 I=IP+1
 1501 J=JP
      IF(I.GT.8) GO TO 1530
      ASSIGN 1510 TO MAR
      GO TO 1590
 1510 J=JP-1
      IF(J.LE.0) GO TO 1520
      ASSIGN 1520 TO MAR
      GO TO 1590
 1520 J=JP+1
      IF(J.GT.8) GO TO 1530
      ASSIGN 1530 TO MAR
```

```
      GO TO 1590
 1530 IF(I.LT.IP) GO TO 1540
      I=IP-1
      IF(I.LE.0) GO TO 1540
      GO TO 1501
 1540 I=IP
      J=JP-1
      IF(J.LE.0) GO TO 1550
      ASSIGN 1550 TO MAR
      GO TO 1590
 1550 J=JP+1
      IF(J.GT.8) GO TO 2000
      ASSIGN 2000 TO MAR
 1590 IF(OCCSQ(I,J).GT.0) GO TO 1595
      N=N+1
      MPOSMV(N)=100*K+10*I+J
 1595 MATTSQ(I,J)=MATTSQ(I,J)+100*JVALPC(K)+1
      MSQ=10*I+J
      IF(MSQ.EQ.KINGSQ) GO TO 2400
      GO TO MAR,(1510,1520,1530,1550,2000)
 2000 CONTINUE
      NT=N
      NPHT=NN
      GO TO 2450
 2400 NCHK=1
EN PASSANT MOVES
 2450 IF(OENP.EQ.0) GO TO 3500
      OPN1=OENP/100
      OKL=OENP-100*OPN1
      IPN=OKL/10
      JPN=OKL-10*IPN
      J=JPN-1
 3010 IF(J.LE.0) GO TO 3060
      IF(J.GT.8) GO TO 3500
      OCSQ=OCCSQ(IPN,J)*NTURN
      IF(OCSQ.NE.10) GO TO 3060
      OQ1=10*IPN+J
      IF(NTURN.EQ.-1) GO TO 3040
      L=0
      DO 3020 K=1,16
      IF(MPC(K).EQ.OQ1) L=K
 3020 CONTINUE
      NT=NT+1
      MPOSMV(NT)=100*L+10*(IPN+1)+JPN
      MATTSQ(IPN,JPN)=MATTSQ(IPN,JPN)+100*JVALPC(L)+1
      GO TO 3060
 3040 L=0
      DO 3050 K=1,16
      IF(OPC(K).EQ.OQ1) L=K
 3050 CONTINUE
      NT=NT+1
      MPOSMV(NT)=100*L+10*(IPN-1)+JPN
      MATTSQ(IPN,JPN)=MATTSQ(IPN,JPN)+100*JVALPC(L)+1
 3060 IF(J.GT.JPN) GO TO 3500
      J=JPN+1
      GO TO 3010
 3500 MNT=NT
      MNPHT=NPHT
      DO 2100 I=1,8
      DO 2100 J=1,8
      OCCSQ(I,J)=OCCSQ(I,J)*NTURN
 2100 CONTINUE
      RETURN
      END
PROGRAM PRO
      DIMENSION NWT(18),NPAR(18)
      READ(8,100) (NWT(K),K=1,18)
  100 FORMAT(18I4)
      READ(8,100) (NPAR(K),K=1,18)
      READ(8,200) LA
  200 FORMAT(I4)
      READ(8,200) LE
```

```
      READ(8,200) N
      DO 300 K=1,18
      NWT(K)=10*NWT(K)+(LA*LE*NPAR(K)*10)/(N+1)
      NWT(K)=NWT(K)/10
300   CONTINUE
      WRITE(6,400) (NS
      WRITE(6,400) (NWT(K),K=1,18)
400   FORMAT(2X,18I4)
      SR
      STOP
      END
```

LIST OF REFERENCES

1. Slagle, J. R., Artificial Intelligence: The Heuristic Programming Approach, p. 4, McGraw-Hill, 1971.

2. Botvinnik, M. M., Computers, Chess and Long-Range Planning, p. 78, English Universities Press Ltd., 1970.

3. Newell, A., Shaw, J. C. and Simon, H. A., "Chess-Playing Programs and the Problem of Complexity," p. 47, in Computers and Thought, Ed. by Feigenbaum, E. A., McGraw-Hill, 1963.

4. Rushton, P. G. and Marsland, T. A., "Current Chess Programs: A Summary of Their Potential and Limitations," INFOR, v. 11, pp. 13-19, February 1973.

5. Slagle, J. R., Artificial Intelligence: The Heuristic Programming Approach, p. 4, McGraw-Hill, 1971.

6. *Ibid.*, p. 3.

7. Nagy, G., "The State of Art in Pattern Recognition," Proceedings of the IEEE, v. 56, pp. 836-862, May 1968.

8. Widrow, B., Gupta, N. K. and Maitra, S., "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems," IEEE Transactions on Systems, Man and Cybernetics, v. SMC-3, p. 456, September 1973.

9. Lewis, P. M. and Coates, C. L., Threshold Logic, pp. 409-417, John Wiley and Sons, 1967.

10. Nagumo, J. I. and Noda, A., "A Learning Method for System Identification," IEEE Transactions on Automatic Controls, v. AC-12, pp. 282-287, June 1967.

11. Widrow, B., Gupta, N. K. and Maitra, S., "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems," IEEE Transactions on Systems, Man and Cybernetics, v. SMC-3, pp. 455-465, September 1973.

12. Simon, H. A. and Chase, W. G., "Skill in Chess," American Scientist, v. 61, p. 394, July-August 1973.

13. Newell, A., Shaw, J. C. and Simon, H. A., "Chess-Playing Programs and the Problem of Complexity," IBM Journal of Research and Development, v. 2, p. 320, October 1958.

14. Kozdrowicki, E. W. and Cooper, D. W., "When Will A Computer be World Chess Champion?," Computer Decisions, v. 6, p. 29, August 1974.

15. Young, J. F., Cybernetics, pp. 1-19, London ILIFFE Books Ltd., 1969.

16. Miller, G. A., "The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information," Psychological Review, v. 63, pp. 81-97, March 1956.

17. Horowitz, I. A. and Reinfeld, F., How to Think Ahead in Chess, p. xi, Simon and Shuster, 1956.

18. Newell, A., Shaw, J. C. and Simon, H. A., "Chess-Playing Programs and the Problem of Complexity," IBM Journal of Research and Development, v. 2, p. 324, October 1958.

19. Shannon, C. E., "Programming a Computer for Playing Chess," The Philosophical Magazine, v. 41, pp. 256-275, March 1950.

20. Newell, A., Shaw, J. C. and Simon, H. A., "Chess-Playing Programs and the Problem of Complexity," IBM Journal of Research and Development, v. 2, p. 334, October 1958.

21. Greenblatt, R. D., Eastlake, D. E. and Crocker, S. D., "The Greenblatt Chess Program," Proceedings of the Association for Information Processing Systems 1967 Fall Joint Computer Conference, v. 31, pp. 801-810.

22. Rushton, P. G. and Marshland, T. A., "Current Chess Programs: A Summary of Their Potential and Limitations," INFOR, v. 11, pp. 14-16, February 1973.

23. *Ibid.*, pp 17-18.

24. Kozdrowicki, E. W. and Cooper, D. W., "COKO III: The Cooper-Koz Chess Program," Communications of the ACM, v. 16, pp. 411-427, July 1973.

25. Simon, H. A. and Chase, W. G., "Skill in Chess," American Scientist, v. 61, pp. 394-397, July-August 1974.

26. Kozdrowicki, E. W. and Cooper, D. W., "COKO III: The Cooper-Koz Chess Program," Communications of the ACM, v. 16, p. 411, July 1973.

27. Botvinnik, M. M., Computers, Chess and Long-Range Planning, pp. 11-25, English Universities Press Ltd., 1970.

28. Shannon, C. E., "Programming a Computer for Playing Chess," The Philosophical Magazine, v. 41, p. 264, March 1950.

29. Good, I. J., "A Five Year Plan for Automatic Chess," p. 96 in Machine Intelligence II, Ed. by Dale, E. and Michie, D., American Elsevier Publishing Co., 1968.

30. Kozdrowicki, E. W. and Cooper, D. W., "When Will A Computer be World Chess Champion?," Computer Decisions, v. 6, p. 32, August 1974.

31. Widrow, B., Gupta, N. K. and Maitra, S., "Punish/Reward: Learning with a Critic in Adaptive Threshold Systems," IEEE Transactions on Systems, Man and Cybernetics, v. SMC-3, p. 456, September 1973.

32. Reinfeld, F., Great Brilliance Prize Games of the Chess Masters, p. 18, Collier Books, 1961.

33. Reinfeld, F., The Complete Chessplayer, p. 138, Fawcett, 1953.

34. *Ibid.*, p. 143.

35. Reinfeld, F., Great Brilliancy Prize Games of the Chess Masters, p. 145, Collier Books, 1961.

36. Reinfeld, F., The Complete Chessplayer, p. 145, Fawcett, 1953.

37. *Ibid.*, p. 148.

38. Reinfeld, F., Great Brilliancy Prize Games of the Chess Masters, p. 88, Collier Books, 1961.

39. *Ibid.*, p. 17.

40. *Ibid.*, p. 24.

41. *Ibid.*, p. 41.

42. *Ibid.*, p. 152.

43. *Ibid.*, p. 152.

44. *Ibid.*, p. 26.

45. *Ibid.*, p. 34.

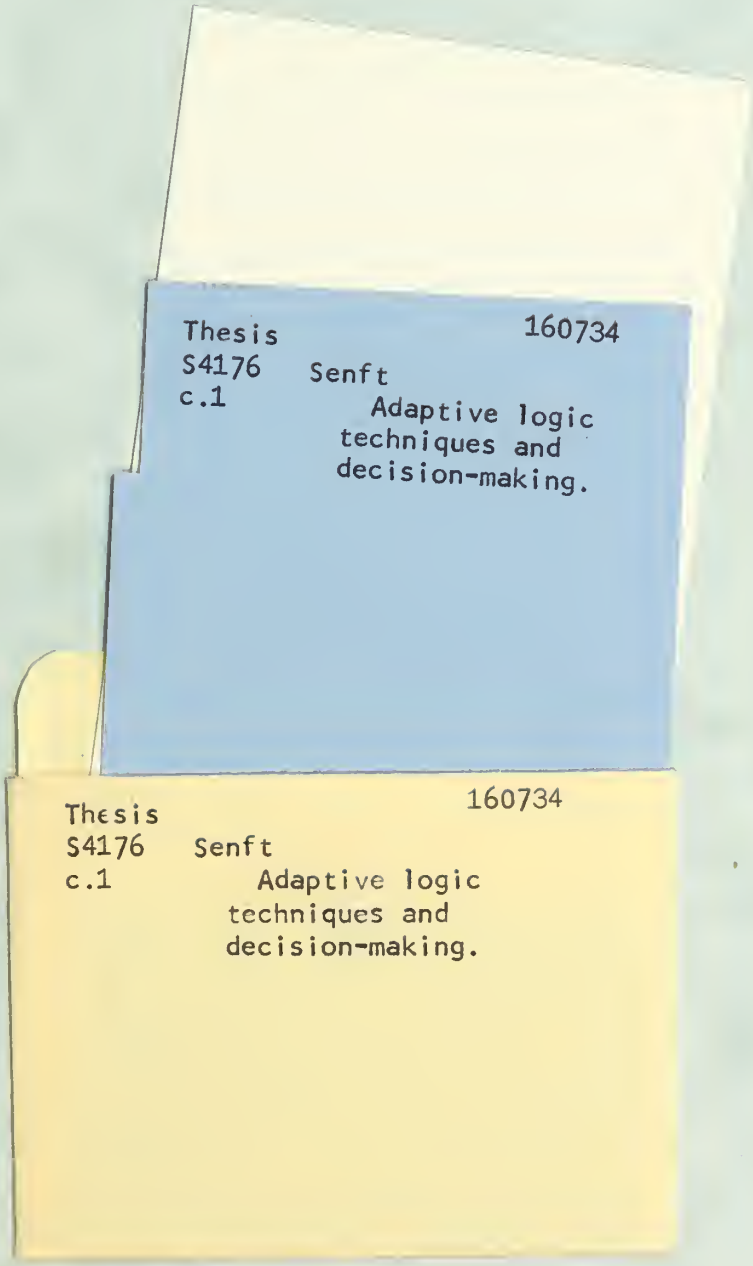
46 *Ibid.*, p. 120.

47. Fischer, R., Bobby Fischer Teaches Chess, p. 319, Bantam, 1966.

48. *Ibid.*, p. 326.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4.	Professor R. Panholzer Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
5.	Professor V. M. Powers Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
6.	LT Milton Leroy Senft Route 156 Pylesville, Maryland 21132	1



Thesis
S4176
c.1
Senft
Adaptive logic techniques and decision-making.
160734